# Clube Delphi

Ano 9 :: Edição 131 :: R$ 11,90

DevMedia

WPF
Aplicações Desktop no .NET

Lookup Personalizado
Construindo um componente para substituir os tradicionais lookups

# Delphi e Silverlight

## Aplicações RIA no Prism XE

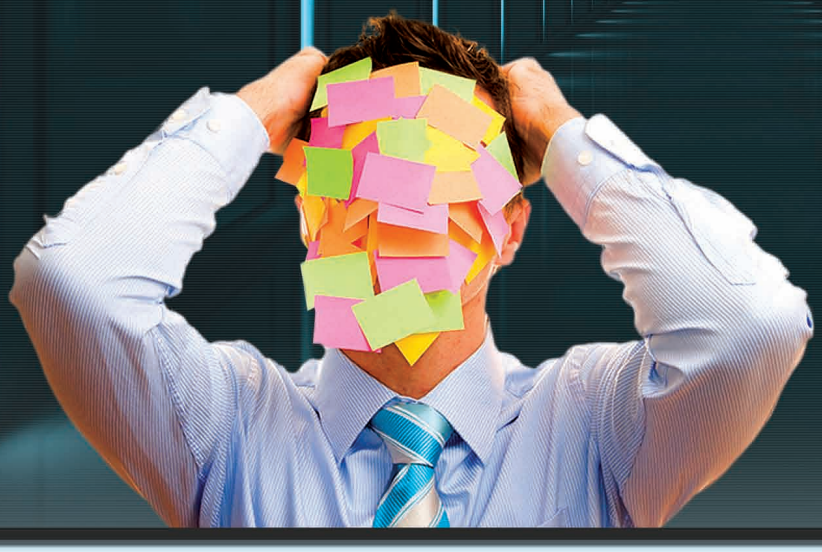
Delphi Prism:
XML, Web Services, LINQ, Lambda e ADO.NET

Bancos de Dados Gratuitos:
Firebird, MySQL e PostgreSQL com Delphi

ISSN 1517990-7

9 771517 990009 001131

# Dificuldades em encontrar conteúdo sobre .Net para iniciantes?



## A DevMedia simplifica pra você!

easy .net magazine

### Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

### Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

**A Easy.Net não é feita em PDF ou Flash!**
**Ela utiliza um aplicativo 100% online.**

DevMedia group

**Acesse agora a edição Nº 1 totalmente grátis e comprove**

**www.devmedia.com.br/easynet**

**Easy.Net Magazine, seu curso inicial de .Net!**

# Sumário


Delphi

.Net

**05 – Introdução ao Silverlight**
Aplicações RIA no Delphi Prism XE
[ *Fabrício Hissao Kawata* ]

.Net

**12 – Introdução ao WPF**
Aplicações Windows Presentation Foundation no Delphi Prism XE
[ *Fabrício Hissao Kawata* ]

Banco de Dados

**19 – Bancos de Dados Gratuitos**
Componentes de conexão
[ *Vitor Luiz Rubio* ]

Boas Práticas

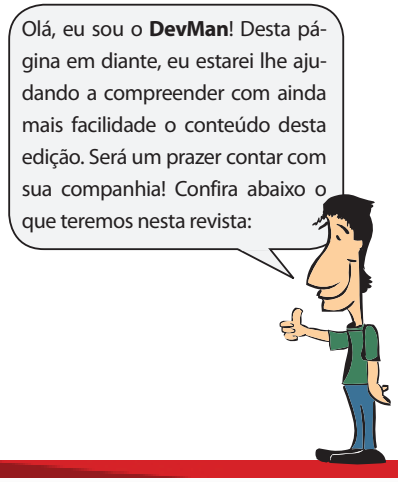
Boas Práticas – Mão na Massa – Boa Ideia

**33 – Lookup Personalizado**
Construindo um componente para substituir os tradicionais lookups
[ *Vitor Luiz Rubio* ]

Easy Delphi

.net - Mão na Massa

**44 – Delphi Prism**
XML, Web Services, LINQ, Lambda e ADO.NET
[ *Rodrigo Araujo* ]

## Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
*www.devmedia.com.br/clubedelphi/feedback*


RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

Clube Delphi

Ano 9 - 131ª Edição 2011 - ISSN 1517990-7 Impresso no Brasil

## Corpo Editorial

**Editor Geral**
Guinther Pauli
guinther@devmedia.com.br

**Editor Técnico**
Paulo Quicoli
pauloquicoli@devmedia.com.br

Leandro Daniel
contato@leandrodaniel.com

**Equipe Editorial**
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

**Jornalista Responsável**
Kaline Dolabella - JP24185

**Atendimento ao Leitor**
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

*www.devmedia.com.br/mancad*
(21) 3382-5038

**Publicidade**
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

**Distribuição**
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

**Fale com o Editor!**

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

**Guinther Pauli - Editor da Revista**
*guinther@devmedia.com.br*

## EDITORIAL

Esta edição destaca o acesso a dados no Delphi Prism. Só para se ter uma ideia, existem dezenas de formas de acesso a dados no .NET Framework, motivo que causa muita confusão, principalmente para quem está iniciando na plataforma. Entre as formas de acesso, podemos citar o uso de ADO.NET através de classes como SqlConnection e SqlCommand (no caso do SQL Server), ou ainda usar o ADO.NET em uma camada separada, como um Data Access Layer, usando boas práticas e separação de responsabilidades, isolando o domínio. DataSets tipados e TableAdapters também são uma boa forma de acesso a dados, pois permitem separar o acesso em um layer independente da UI. Usando o ASP.NET temos o modelo baseado em DataSource, que tira proveito do mecanismo de DataBinding, incluindo o SqlDataSource e ObjectDataSource. Para um desenvolvimento mais profissional e orientado a objetos, podemos usar um framework como o NHibernate ou o ADO.NET Entity Framework.

O ADO.NET Entity Framework (EF) é a ferramenta de persistência e mapeamento objeto / relacional nativa do .NET Framework, concorrente direto do NHibernate. Ele originalmente apareceu pela primeira vez no Service Pack 1 do Visual Studio 2008 / .NET 3.5. No Visual Studio 2010 / .NET 4.0 ele está na sua versão 2, apesar de normalmente nos referenciarmos a ele por EF4 (versão 4). O ADO.NET EF permite o desenvolvimento de aplicações para .NET onde o foco principal é a orientação a objetos, sem uma comunicação direta com um servidor de banco de dados relacional, como o SQL Server. É o framework que gera todos os comandos SQL. A última versão permite que o desenvolvedor inicie a criação de um aplicativo pela definição do domínio (entidades), com um diagrama de classes, podendo a seguir gerar todo o esquema para o SGBDR. Essa técnica é conhecida como Model First Development. Se já possui um banco existente, o EF pode fazer a engenharia reversa, gerando entidades no modelo a partir do BD. Outra característica interessante do EF é a capacidade de suportar tipos complexos, uma novidade da última versão, bem como Lazy Loading, que permite o carregamento de entidades relacionadas conforme utilização. O que considero um dos melhores recursos é o novo suporte a POCO (Plain Old CLR Objects), onde podemos modelar o domínio da aplicação, com as entidades, sem gerar código que esteja atrelado ao framework ou a uma tecnologia em específico. As classes do domínio simplesmente mapeiam a tabela do banco, normalmente sem conter métodos ou derivar de uma classe base para que possam funcionar. Isso dá uma maior flexibilidade e independência ao se trabalhar com o EF. Isso é importante para aplicações em camadas, que prezam por baixo acoplamento entre suas partes, seguindo bons princípios do DDD (Domain Driven Design). Nesta edição, damos especial destaque ao uso dessa poderosa ferramenta para acesso a dados, utilizando o Delphi Prism XE.

**Guinther Pauli**
*guinther@devmedia.com.br*
Certificado Delphi 3,5,6,7,2005,2006,Web,Linux
Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD (ASP.NET,C#)
guintherpauli.blogspot.com - twitter.com/guintherpauli

ClubeDelphi PLUS

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
*http://www.devmedia.com.br/clubedelphi/portal.asp*

DevMedia

# Introdução ao Silverlight

## Aplicações RIA no Delphi Prism XE

Fabrício Hissao Kawata

A evolução da área de desenvolvimento tecnológico como um todo, culminou com uma grande abrangência de opções de trabalho. Em outras palavras, hoje são inúmeras as opções de linguagens, tecnologias, ferramentas, padrões, entre outras. Com tudo isso, o nível das aplicações, de um modo geral, também cresceu, assim como as necessidades e exigências dos usuários. Hoje, eles não querem só um sistema com as funcionalidades necessárias para o seu negócio, mas também que o aplicativo ofereça um visual rico de forma a aumentar sua experiência, além de poder utilizá-lo a partir de qualquer lugar do globo.

Parte disto é facilmente conseguido com as aplicações Desktop tradicional que, através de suas interfaces de usuário, fornecem uma devida interação com quem as utilizam. No entanto, nem todas as aplicações deste tipo estão aptas a serem acessadas remotamente. Eis que surgem então as aplicações Web, que atendem bem a este quesito, porém esbarram na questão visual e funcional. Isto se deve a questões como diferença de plataforma, de recursos disponíveis e da dependência de um conhecimento em relação a design para a elaboração de um visual melhor, que certamente não é o forte da maioria dos desenvolvedores.

### RIA (Rich Internet Applications)

RIA é a sigla para Rich Internet Applications que, em português, pode ser traduzido como Aplicações de Internet Rica. Este termo, aliás, foi utilizado pela primeira vez em 2002 pela Macromedia e pode ser taxado como uma referência às aplicações Web que apresentam características e funcionalidades das aplicações Desktop. Este "novo" tipo de aplicação surgiu para preencher a lacuna que existia entre as soluções Desktop e Web tradicionais. Enquanto a primeira apresentava como características ser "rica" e de baixo alcance, a outra era "pobre", porém de longo alcance. Entenda como baixo alcance o fato de uma aplicação só estar disponível somente nas máquinas onde foi realizada sua instalação e longo alcance, quando se tem a aplicação publicada uma única vez no servidor, estando acessível a todos que o acessarem. O conceito RIA visa à mescla desses dois contextos, obtendo o melhor de ambos: uma aplicação rica e de longo alcance.

Há inúmeras soluções RIA disponíveis atualmente no mercado (como Adobe Flash, Adobe Flex e o Microsoft Silverlight, só para citar alguns dos mais conhecidos), todavia, uma característica pertinente a todas é o envolvimento de um plug-in específico da tecnologia na máquina cliente. Em tempo de execução (runtime) o plug-in fará o intermédio entre o servidor e o usuário.


## Resumo DevMan

***De que trata o artigo:***

O presente artigo abordará de forma introdutória o Silverlight, a plataforma RIA (Rich Internet Application) da Microsoft. Para isso, os conceitos primordiais sobre a tecnologia serão passados ao leitor, proporcionando, dessa forma, mais uma opção de desenvolvimento para a comunidade Delphi. A integração destas duas tecnologias é devidamente conseguida graças ao Delphi Prism, que tem como base o Visual Studio, o principal IDE da Microsoft.

***Para que serve:***

RIA (Rich Internet Applications) é a nomenclatura imposta às soluções Web que proporcionam ricas experiências ao usuário. Este tipo de aplicação tem, dentre outras características, um visual atraente e funcionalidades tão boas quanto às das aplicações Desktop. Além disso, são executadas na máquina cliente, sendo a ligação entre o usuário e o servidor. Neste cenário, o Silverlight surge como a tecnologia da Microsoft para estes tipos de soluções.

***Em que situação o tema é útil:***

Aplicações Silverlight cumprem bem seu papel RIA, proporcionando excelente experiência ao usuário através de um aplicativo Web, com um grau de agilidade semelhante a uma aplicação Desktop. Somado a isso, o Silverlight pode rodar em plataformas (Windows, Mac e outros) e navegadores Web diferentes (Internet Explorer, Firefox, Safari e outros).

***Introdução ao Silverlight:***

A pretensão do artigo é de apresentar a tecnologia Silverlight, a solução RIA da Microsoft, de forma a deixar o leitor confortável em escolher aplicações deste tipo para o desenvolvimento de novos projetos com o Delphi Prism. Para atingir este objetivo, será apresentada uma introdução conceitual ao assunto, para que o leitor adquira o embasamento necessário para dar continuidade aos seus estudos sobre a tecnologia apresentada, visto a extensão e complexidade da mesma. Por fim, com o conhecimento geral do Silverlight em mente, o desenvolvedor poderá mensurar a utilidade desta plataforma em seus projetos, sendo capaz, por exemplo, de oferecer uma aplicação para internet tão funcional quanto uma Desktop tradicional.


### Silverlight

O Silverlight, apresentado inicialmente como WPF/E (Windows Presentation Foundation Everywhere), pode ser definido como uma tecnologia, criada pela Microsoft, de características cross-

platform e cross-browser, que permite a criação de ricos e interativos aplicativos para Web, as já citadas Rich Internet Applications (RIA). Logo, obedecendo à regra, toda a linha de execução do Silverlight se dá basicamente na máquina cliente, sendo instalado no ambiente do usuário na forma de um plug-in.

Nesse contexto, podemos dizer que um Plug-in, add-in ou add-on, é a definição para um pequeno programa de computador que acrescenta funcionalidades específicas a outros programas maiores. Já o termo cross-platform, também popularmente conhecido como mult-platform (multi-plataforma), refere-se ao item computacional elaborado para atuar em várias e distintas plataformas de computador. Tome como exemplo uma aplicação multi-plataforma que pode ser executada em um ambiente Microsoft Windows, Linux ou Mac OS X.

Já o termo cross-browser faz referência à capacidade de um item Web (aplicação, site, script, etc) suportar distintos navegadores de internet. Em outras palavras, independente do navegador, o item deve ser devidamente interpretado, executado ou exibido, atingindo um resultado final ideal.

As tecnologias voltadas para o conceito RIA por si só apresentam características já mencionadas que justificam sua utilização, e com o Silverlight não é diferente. Contudo, além dessas, ainda possui outras características peculiares que o diferenciam dentre as demais, como veremos a seguir.

## Cross-browser/Cross-Platform

Pode-se citar como exemplo das inconveniências encontradas no mundo Web, a alteração de comportamento de uma página Web tradicional, ocasionada pela simples mudança do navegador que está sendo utilizado para executá-la. Talvez um simples script encontrado nesta página se comporte diferente, devido ao fato de cada browser interpretá-lo de maneira distinta. Em outro caso, uma funcionalidade de uma aplicação ASP.NET pode ter seu comportamento diferenciado dependendo da versão do Internet Explorer.

Esse tipo de problema é suprido com a tecnologia RIA da Microsoft, através de sua característica cross-browser. Ao desenvolver uma aplicação Silverlight, tem-se a certeza de que ela irá rodar e se comportar exatamente igual em todos os navegadores suportados, tais como: Internet Explorer, Google Chrome, Firefox e Safari.

Mantendo essa linha de conceito, o Silverlight também é apresentado como sendo cross-platform, ou seja, é possível executá-lo em múltiplas e diferentes plataformas. Nativamente são duas as suportadas: as baseadas em Windows e Mac OS. Além disso, existe um projeto paralelo, fruto de uma parceria entre Microsoft e Novell, denominado Moonlight, que é uma implementação open-source do Silverlight para sistemas operacionais baseados em Linux e Unix/X11.

## Integração com linguagens .NET

A partir de sua versão de número 2, o Silverlight passou a possuir uma versão própria e reduzida do .NET Framework que implementa também uma versão exclusiva da CLR, contando com uma biblioteca de objetos próprios, possibilitando assim a escrita de programas em qualquer linguagem .NET (e neste grupo se encaixa a linguagem Oxygene do Delphi Prism). Na maioria dos casos, a linguagem em questão é utilizada para a escrita da lógica de programação referente às regras de negócio da aplicação que está se desenvolvendo. No entanto, é possível também, por exemplo, manipular via código os controles Silverlight.

CLR (Common Language Runtime) pode ser interpretada popularmente como o coração da plataforma .NET. Em termos técnicos, pode ser definida como sendo uma "máquina virtual" responsável por providenciar a execução de aplicações escritas nas linguagens .NET, garantindo uma interoperabilidade entre elas.

## Conhecimento único para o ambiente de desenvolvimento

O instalador do Delphi Prism possui um shell do Visual Studio, que é como uma versão totalmente básica do software. Ele é instalado nos ambientes em que o IDE da Microsoft não está presente, do contrário, o Delphi Prism se integra ao Visual Studio em forma de plug-in. Toda esta citação tem como objetivo explicitar que para o desenvolvimento com o Silverlight, não é necessário o desenvolvedor focar no aprendizado de outro IDE. Somente o IDE do Delphi Prism é o suficiente, ou seja, todo seu conhecimento relativo à ferramenta é aproveitado.

Claro que existem outras ferramentas externas, não obrigatórias, que podem agregar produtividade a esse ambiente de desenvolvimento, como é o caso do Expression Blend. Um software da Microsoft voltado para a parte de design que, no entanto, é capaz de criar e editar aplicações Silverlight, da mesma forma que no Visual Studio, em relação à parte visual dos projetos. Por serem da mesma fabricante, a integração entre as duas ferramentas é perfeita e o objetivo de otimização de produtividade é facilmente atingido.

### Nota do DevMan

Oxygene, anteriormente conhecida como Chrome, é uma linguagem de programação baseada em Object Pascal desenvolvida pela empresa RemObjects Software, voltada para o desenvolvimento .NET. A partir de 2008 licenciou seus produtos (compilador e IDE) para a Embarcadero, resultando na IDE Delphi Prism, que se integra devidamente ao Visual Studio, aproveitando todos os recursos oferecidos pela tecnologia .NET.

### Nota do DevMan

Microsoft Expression Blend é uma ferramenta voltada para interfaces de usuário, sendo utilizada para a criação de interfaces gráficas tanto para aplicações Web quanto para aplicações Desktop, atendendo dessa forma projetos WPF (Windows Presentation Foundation) e Silverlight, respectivamente. Ele é membro integrante da família Expression da Microsoft, que é representada pelo Microsoft Expression Studio, uma suíte de aplicativos voltados principalmente para multimídia. A concepção das interfaces gráficas criadas com o Expression Blend se dá por meio do contexto WYSIWYG (What You See Is What You Get - O que você vê é o que você obtém) resultando em código XAML.

## Requisito mínimo para execução

Para que uma aplicação Silverlight possa ser devidamente executada em uma máquina, basta que seu client runtime esteja instalado na estação cliente. Entenda por client runtime do Silverlight como sendo um plug-in gratuito compatível com os navegadores, dispositivos e sistemas operacionais suportados, que proporcionará todas as funcionalidades necessárias ao ambiente integrado à Web. Neste quesito, a Microsoft fez todo seu trabalho para tornar o referido plug-in o mais acessível a todos, de modo a deixá-lo o mais simples e leve possível. Desde o lançamento da tecnologia, a Microsoft obteve o sucesso almejado, tanto que atualmente, em sua versão estável mais recente (até o fechamento deste artigo) de número 4, o plug-in do Silverlight possui aproximadamente meros 6MB.

Não obstante, caso o runtime não esteja presente na máquina cliente, ao tentar se executar uma aplicação Silverlight no navegador, uma agradável mensagem é mostrada ao usuário, instruindo-o a baixar o plug-in, conforme mostra a **Figura 1**.

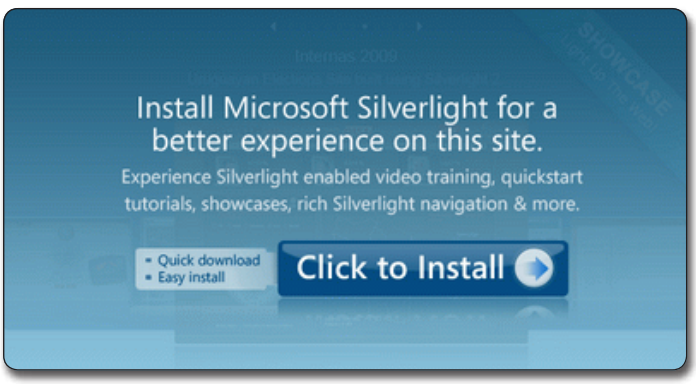



**Figura 1.** Aviso para instalação do plug-in Silverlight

## Principal plataforma para desenvolvimento com Windows Phone 7

Ao falar das características e vantagens proporcionadas pelo desenvolvimento Silverlight, atualmente é impossível não fazer menção ao seu suporte ao Windows Phone 7, o novo sistema operacional para dispositivos móveis da Microsoft. O Silverlight é simplesmente a opção principal para desenvolvimento de aplicações para o Windows Phone, possuindo uma versão exclusiva para o mesmo. Apesar de esta ser tratada como "Silverlight for Windows Phone", a versão faz uso da mesma biblioteca de classes base de sua versão tradicional. Isso permite que todo o seu conhecimento adquirido sobre a tecnologia seja eficazmente aproveitado na plataforma móvel.

## XAML

O Silverlight utiliza, para a definição de suas interfaces de usuário, uma linguagem de marcação essencialmente em texto, baseada em XML. Ela foi criada pela Microsoft e é denominada XAML, um acrônimo de eXtensible Application Markup Language. É tão importante e essencial ao Silverlight como o HTML é para o desenvolvimento Web. Na **Listagem 1** a seguir é mostrado um exemplo de declaração XAML, que é apresentado em um projeto Silverlight no Delphi Prism.

A definição de uma interface de usuário com XAML é devidamente suportada pelo IDE (Delphi Prism/Visual Studio) e podendo se dar de duas formas: via código ou através do tradicional arrastar-e-soltar de componentes. Com relação a este último, a IDE pode até ser considerada uma excelente ferramenta WYSIWYG, oferecendo em sua Toolbox uma gama de controles para a tecnologia (**Figura 2**). Ao mesmo tempo, a IDE nos oferece a manipulação das propriedades e eventos, de modo visual, através da janela de editor de propriedades (**Figura 3**). No entanto, o resultado obtido por estes processos pode nem sempre produzir o XAML mais adequado e eficiente para a aplicação, daí a importância em se aprender os detalhes da linguagem de marcação utilizada pelo Silverlight, para o manuseio manual do código, quando necessário.

Ainda sobre a linguagem, os elementos e seus atributos declarados em XAML resultam em objetos em memória, portanto, cada elemento pode ser interpretado como a instância de uma determinada classe. Veja a seguir na **Listagem 2** um exemplo simples da declaração de duas instâncias, cada qual com seus atributos: um Button, da classe System.Windows.Controls.Button e um TextBox, da classe System.Windows.Controls.TextBox.

## Primeira aplicação prática

Visto a teoria, nada melhor que uma abordagem mais prática para que os conceitos sejam realmente fixados. Ainda que o escopo do artigo não seja criar uma aplicação Silverlight do início ao fim, serão demonstrados a seguir os detalhes para a construção de uma primeira aplicação com o Silverlight no Delphi Prism, no mais tradicional estilo "Hello World".

Na IDE do Delphi Prism/Visual Studio, uma aplicação Silverlight é construída através de tipos de projetos específicos à tecnologia, conforme pode ser visto na **Figura 4**. Para uma primeira experiência prática, você pode criar um novo projeto utilizando o modelo Silverlight Application que é a opção mais simplista para se construir uma aplicação Silverlight.

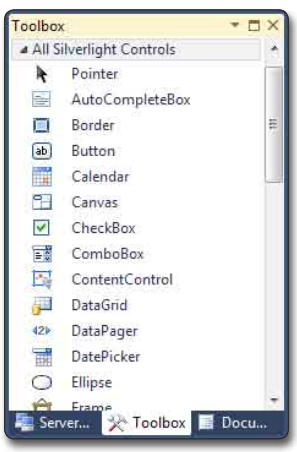



**Figura 2.** Toolbox com controles Silverlight

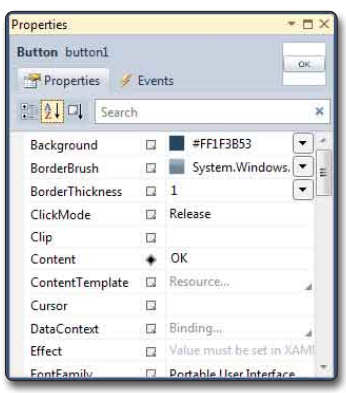



**Figura 3.** Editor de propriedades dos controles Silverlight

Ao selecionar este tipo, tenha em mente que o aplicativo Silverlight precisará ser hospedado em uma página Web que rodará no browser, logo, a própria IDE já oferece algumas opções para esta hospedagem. Para isso, uma nova tela é apresentada, conforme mostra a **Figura 5**. Neste momento, é possível escolher o tipo de projeto Web, dentre os quais pode ser, por exemplo, um ASP.NET Web Application.

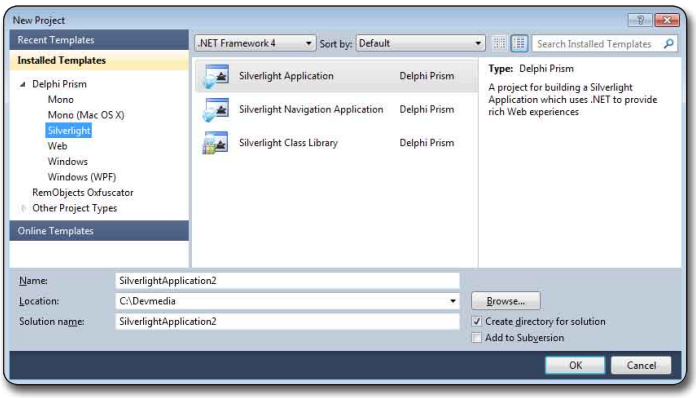



**Figura 4.** Tipos de projetos Silverlight no Delphi Prism

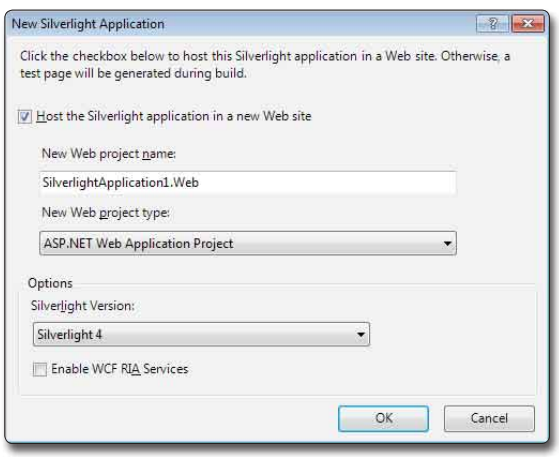



**Figura 5.** Selecionado o Host para o projeto Silverlight

Feita a escolha, a IDE cria automaticamente um solução base para você, contendo os dois projetos citados (**Figura 6**).

SilverlightApplication1 é o projeto Silverlight em si. Por padrão, inicialmente contém dois arquivos XAML: App.xaml e Page.xaml. O primeiro, App.xaml, pode ser considerado o ponto central da aplicação, contendo, entre outras coisas, a declaração de seus recursos compartilhados. Além disso, ao criar este arquivo, a IDE declara em seu code-behind (o arquivo .pas contendo o código por trás do .xaml) alguns manipuladores de eventos pré-definidos, em nível global da aplicação, conforme mostra a **Listagem 3**. Tais métodos já trazem em seu corpo alguma implementação default, mas nada impede que você as adeque para o seu contexto. Já o arquivo Page.xaml pode ser definido como a página inicial da aplicação, aquela que irá ser mostrada ao usuário no momento da execução do aplicativo. Seu conteúdo XAML padrão pode ser visto na **Listagem 4** e ela está apta a receber os elementos visuais que irão compor sua interface de usuário.

**Listagem 1.** Exemplo de código XAML

```
<?xml version='1.0' encoding='utf-8' ?>
<UserControl x:Class="SilverlightApplication3.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="400">
  <Grid x:Name="LayoutRoot" Background="White">
  </Grid>
</UserControl>
```

**Listagem 2.** Exemplo simples de código XAML

```
<Grid x:Name="LayoutRoot" Background="White">
<Button Name="button1" Content="OK"
        Height="23" Width="58"
        VerticalAlignment="Top" HorizontalAlignment="Right"
        Margin="0,27,190,0" />
   <TextBox Name="textBox1"
         Height="23" Width="120"
         VerticalAlignment="Top" HorizontalAlignment="Left"
         Margin="26,27,0,0" />
</Grid>
```

**Listagem 3.** Eventos declarados no code-behind de App.xaml

```
App = public partial class(System.Windows.Application)
 private
  method ReportErrorToDOM(e: ApplicationUnhandledExceptionEventArgs);
  method Application_Startup(sender: object; e: StartupEventArgs);
  method Application_Exit(sender: object; e: EventArgs);
  method Application_UnhandledException(sender: object;
      e: ApplicationUnhandledExceptionEventArgs);
 public
  constructor;
 end;
```

**Listagem 4.** Conteúdo XAML de Page.xaml

```
<?xml version='1.0' encoding='utf-8' ?>
<UserControl x:Class="SilverlightApplication3.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"
  d:DesignHeight="300" d:DesignWidth="400">
  <Grid x:Name="LayoutRoot" Background="White">
  </Grid>
</UserControl>
```

SilverlightApplication1.Web é o projeto Web que é associado ao projeto Silverlight. Como já foi dito, esta aplicação tem como objetivo hospedar o aplicativo Silverlight de modo a embuti-lo em uma página Web. Por isso, a IDE constrói o projeto já pronto para ser executado. Isso quer dizer que todas suas configurações já estão devidamente direcionadas à aplicação Silverlight.

Faça o teste e apenas execute o projeto (F5) do jeito que está, sem fazer alteração alguma, e veja que ele é executado normalmente em seu navegador. Ainda sobre o projeto, você pode perceber também que, por se tratar, neste caso, de um ASP.NET Web Application, ele possui todos os arquivos pertinentes a este tipo de projeto, como por exemplo: arquivo de configuração (Web.config), páginas aspx e HTML e arquivos JavaScript.

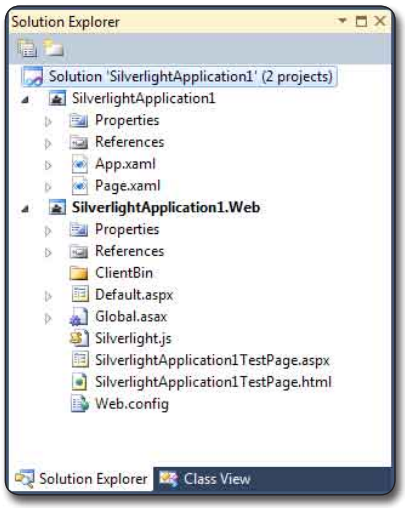



**Figura 6.** Solução criada pelo IDE

## Gerenciamento de Layout

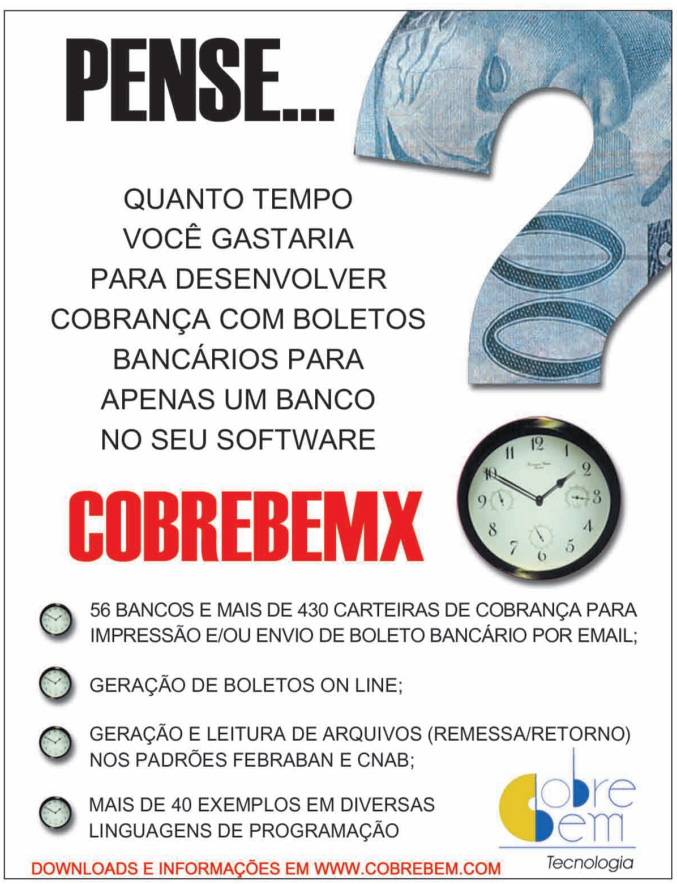
Antes de dar prosseguimento com a construção desta primeira aplicação prática com o Silverlight, é necessário abordar alguns conceitos básicos referente à estrutura do código XAML do arquivo Page.xaml. Portanto, de volta à **Listagem 4**, percebemos que todo o código está envolvido pela declaração da marcação <UserControl>. Esta indicação se refere a uma instância da classe System.Windows.Controls.UserControl e é o controle Silverlight que contém todos os demais elementos que compõem a interface de usuário em um arquivo XAML, sendo assim considerado um controle de alto nível.

Além disso, ao incluir novos elementos visuais é imprescindível a utilização de outro tipo de objeto para gerenciá-los, os chamados controles de layout. Eles são controles que servem de container (painel) para os outros demais elementos visuais, e definem a maneira como os elementos serão dispostos e organizados em tela. O Silverlight oferece cinco opções de objetos que realizam o gerenciamento de layout de uma aplicação:

- Canvas: é um controle de layout básico, talvez o mais simples de todos, baseado na posição absoluta dos controles, ou seja, os objetos são posicionadas através de coordenadas explícitas relativas ao local no painel;
- StackPanel: este tipo de painel dispõe os controles em pilhas horizontais e verticais. Por usar este conceito simplista, é muito utilizado, por exemplo, em aplicações rápidas, nos casos em que a preocupação com a posição dos controles não é tão importante;
- Grid: é o mais flexível dos controles de layout e também o oferecido por padrão pela IDE, conforme pode ser conferido na **Listagem 4**. O posicionamento dos elementos em um Grid pode ser comparado ao posicionamento dos itens em uma tabela HTML, através de linhas e colunas;
- WrapPanel: quase idêntico ao StackPanel, se não fosse pelo fato de que ele reposiciona os elementos para outra faixa de linha ou coluna, nos casos onde há sobra de elementos de uma pilha, pelo fato da altura ou largura do painel já terem sido atingidos;
- DockPanel: trabalha com o encaixe de painéis horizontais e verticais, de forma a preencher todo o espaço disponível no layout.

Dito isto, mantenha a opção de Grid trazida por padrão pelo IDE, no código XAML do arquivo Page.xaml. Dessa forma, já podemos inserir os controles visuais da aplicação de exemplo, que apresentará um simples TextBox e um botão ao usuário. Conforme o botão é pressionado, o texto digitado no controle TextBox é exibido em forma de mensagem. Tudo muito simples. Portanto, decore a página de forma a deixá-la parecida com a **Figura 7**. Para este visual foram utilizados controles simples, presentes na Toolbox da IDE, como Label, TextBox e um Button. O trecho principal XAML é apresentado na **Listagem 5**.

**Figura 7.** Visual da aplicação de exemplo



## Code-behind

Uma vez vista a estrutura do projeto Silverlight criado na Solution Explorer, é possível notar que cada arquivo .xaml possui um arquivo de code-behind associado, que são os de extensão .pas (**Figura 8**). Como já mencionado, neles estarão os códigos de negócio da aplicação. Para elucidar seu uso, utilizaremos a codificação do botão de nosso exemplo.

Portanto, de volta ao arquivo Page.xaml, ao dar um duplo-clique sobre o botão "Mostrar Mensagem" é automaticamente gerado no arquivo de code-behind a declaração do código de seu evento Click, como no desenvolvimento Delphi Win32 tradicional. Ao mesmo tempo, no próprio arquivo XAML é feita uma referência a este evento criado para o botão, dentro das marcações do objeto, da seguinte forma: Click="button1_Click". Através dessa referência, temos uma integração do código XAML com a linguagem utilizada no code-behind. Para terminar, o código do evento OnClick do botão é apresentado na **Listagem 6**.

Ainda sobre esta codificação, repare que na linha 4 foi feita uma menção ao elemento TextBox (textBox1), presente no layout da aplicação. Isso mostra que, no code-behind, também temos acesso aos objetos declarados no arquivo XAML associado.

Ao rodar a aplicação agora, você irá comprovar que ao se digitar um texto qualquer na caixa de entrada e em seguida pressionar o botão Mostrar Mensagem, uma caixa de mensagem será exibida em tela (**Figura 9**), referente à codificação feita no code-behind.

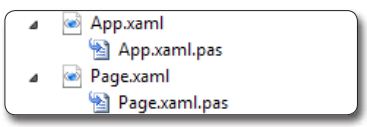



**Figura 8.** Arquivos de code-behind associados aos arquivos XAML

**Figura 9.** Testando a aplicação de exemplo

## Arquivo .xap

Ao compilar um projeto Silverlight ele gera um arquivo .xap com o mesmo nome do projeto, que nada mais é que um arquivo assembly contendo a compilação de todos os arquivos XAML e alguns demais recursos da aplicação. É como se fosse um arquivo ZIP, tanto que se você renomeá-lo para esta extensão, conseguirá visualizar os arquivos presentes em sua estrutura (**Figura 10**). Com base em nosso exemplo, o arquivo XAP gerado pode ser encontrado na pasta do projeto Silverlight, mais especificamente em sua pasta Debug, localizada dentro da pasta Bin.

**Listagem 5.** Código XAML de definição visual da aplicação exemplo

```
<Grid x:Name="LayoutRoot" Background="White">
  <Grid.RowDefinitions>
    <RowDefinition Height="210*" />
    <RowDefinition Height="90*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="400"/>
  </Grid.ColumnDefinitions>
  <TextBox Grid.Row="0" Name="textBox1"
        Height="23" Width="238" Margin="30,88,0,0"
        HorizontalAlignment="Left" VerticalAlignment="Top" />
  <Button Grid.Row="0" Name="button1" Content="Mostrar Mensagem"
       Height="23" Width="114" Margin="0,88,12,0"
       HorizontalAlignment="Right" VerticalAlignment="Top" />
  <sdk:Label Grid.Row="0" Name="label1"
         Content="Aplicação Teste -  Silverlight no Delphi Prism"
         FontWeight="Bold" FontSize="12"
         Height="28" Width="358" Margin="30,22,0,0"
         HorizontalAlignment="Left"  VerticalAlignment="Top" />
</Grid>
```

**Listagem 6.** Codificação do evento Click do botão Mostrar Mensagem

```
01 method MainPage.button1_Click(sender: System.Object;
02 e: System.Windows.RoutedEventArgs);
03 begin
04  MessageBox.Show("Texto digitado: " + textBox1.Text);
05 end;
```

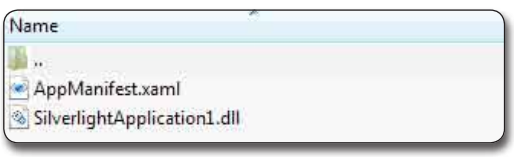



**Figura 10.** Arquivos contidos dentro de um arquivo XAP

Além disso, o arquivo XAP é o arquivo que deve ser referenciado em uma página Web para que ela hospede a aplicação Silverlight. Na solução criada pela IDE, este vínculo pode ser facilmente percebido olhando-se para a estrutura do projeto Web. Ele possui uma pasta ClientBin que é o local onde automaticamente é copiado o arquivo XAP gerado pelo projeto Silverlight, conforme pode ser visto na **Figura 11**. Perceba que a solução criada pela IDE já deixou tudo pronto para o nosso desenvolvimento, fazendo a devida integração entre os dois projetos.

Em termos de código, podemos ver a referência que é feita ao arquivo XAP no corpo da página, cujo arquivo é terminado em TestPage.aspx. Por padrão, esta é a página inicial do projeto Web e, portanto, a que conterá a referência citada. Em seu código HTML pode ser vista a referência que é feita ao arquivo XAP, conforme mostra a **Listagem 7**.

## Conclusão

Vimos a desenvoltura do Silverlight para o desenvolvimento RIA. Poder oferecer ao seu cliente, não só uma aplicação Web eficaz e funcional, nos mesmos moldes de uma aplicação Desktop, como também atraente visualmente falando. Acrescente a isso suas características cross-browser e cross-platform. Temos todas estas novas possibilidades de desenvolvimento do mundo .NET proporcionado pelo Delphi Prism.

**Listagem 7.** Trecho de código HTML que referencia o arquivo XAP

```
01 <object data="data:application/x-silverlight-2,"
02   type="application/x-silverlight-2" width="100%" height="100%">
03   <param name="source" value="ClientBin/SilverlightApplication1.xap"/>
04   <param name="onError" value="onSilverlightError" />
05   <param name="background" value="white" />
06   <param name="minRuntimeVersion" value="4.0.50826.0" />
07   <param name="autoUpgrade" value="true" />
08   <a href="http://go.microsoft.com/fwlink/?LinkID=149156&v=4.0.50826.0"
09        style="text-decoration:none">
10        <img src="http://go.microsoft.com/fwlink/?LinkId=161376" alt="Get
11          Microsoft Silverlight" style="border-style:none"/>
12   </a>
13 </object>
```

**Figura 11.** Arquivo XAP presente no projeto Web

Mas nem tudo são flores. É importante que se tenha em mente que a escolha da tecnologia a ser usada em um projeto deve se adequar às necessidades do negócio. Isso quer dizer que não se deve escolher determinada plataforma somente por ser a "mais legal", ou por ser a "tendência" do momento.

Mais importante que o aprendizado do Silverlight, é possuir uma base coesa tanto da linguagem utilizada pelo Delphi Prism, a Oxygene, como da própria IDE em si. Por isso, como dica, fica a leitura dos inúmeros artigos desta revista, voltados ao assunto, como também os vídeos disponíveis no portal DevMedia.

**Fabrício Hissao Kawata**
*fabricio.kawata@bol.com.br*
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.


**Delphi Prism XE - Página de apresentação do produto**
*http://www.embarcadero.com/products/delphi-prism*

**Delphi Prism XE – Página para download**
*https://downloads.embarcadero.com/free/delphi_prism*

**Microsoft Silverlight - Página oficial**
*http://www.silverlight.net*

**Microsoft Silverlight – Showcase**
*http://www.silverlight.net/showcase*

**Microsoft Silverlight 4 Tools for Visual Studio 2010**
*http://www.microsoft.com/downloads/en/details.aspx?FamilyID=b3deb194-ca86-4fb6-a716-b67c2604a139&displaylang=en*

**Microsoft Expression Blend - Página oficial**
*http://www.microsoft.com/expression/products/Blend_Overview.aspx*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

**Faça um *upgrade* em sua carreira**

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviçoes); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

# Introdução ao WPF

## Aplicações Windows Presentation Foundation no

Paralelamente ao lançamento do Delphi 2009, a Embarcadero apresentou sua nova ferramenta para desenvolvimento com o .NET Framework, denominada Delphi Prism. Na época, tratou-se de uma grande revolução, visto que este novo produto não possuía mais relação alguma com a IDE RAD Studio. Pelo contrário, o Delphi Prism era totalmente baseado no Microsoft Visual Studio, ou melhor dizendo, se integra a ele em forma de plug-in, com o intuito de aproveitar ao máximo os recursos mais recentes do .NET Framework.

Em termos de código, o Prism apresentou o Oxygene como sua linguagem padrão, cuja sintaxe é muita próxima ao Object Pascal do Delphi. Em vista disso, têm-se naturalmente uma curva de aprendizado muito pequena, em relação aos desenvolvedores Delphi que se aventurarem no trabalho com .NET. Desde então, através do Delphi Prism, eles podem facilmente construir qualquer tipo de aplicação baseada no framework. Só para citar alguns destes tipos, é possível ter projetos: ASP.NET, Windows Forms, Web Services, WCF, Silverlight e WPF.

### Windows Presentation Foundation

WPF é o acrônimo de Windows Presentation Foundation, uma API unificada e criada pela Microsoft, cuja principal finalidade é a construção de ricas interfaces gráficas de usuário, as chamadas UI (User Interfaces), para aplicações desenvolvidas com o .NET Framework. Em termos práticos, o WPF mescla características do desenvolvimento Web, RIA e Desktop, tais como o uso de folhas de estilo (CSS – Cascading Style Sheets), utilização de gráficos vetoriais e data binding, respectivamente.

Historicamente falando, o WPF surgiu na época do .NET 3.0, sendo uma de um todo de quatro bibliotecas irmãs apresentadas na época, assim como pode ser visto na **Figura 1**.

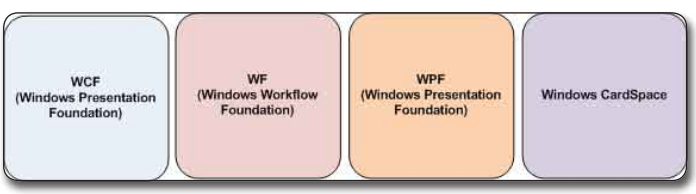



**Figura 1.** Os quatro pilares do .NET Framework 3.0

WCF é a sigla para Windows Communication Foundation, que oferece uma API focada em mensagens, voltada para construção de aplicações orientadas a serviços e computação distribuída. WF é a sigla para Windows Workflow Foundation, cuja API fornece mecanismos para demonstrar e controlar os fluxos de trabalho em um processo. CardSpace fornece um sistema único de identificação para ser usado em sistemas Desktop, sistemas Web, Web Sites e afins.

## Resumo DevMan

***De que trata o artigo:***

O artigo abordará a tecnologia Windows Presentation Foundation (ou simplesmente WPF) da Microsoft. Ela, em conjunto com o .NET Framework, são utilizados na construção de interfaces ricas de usuário, para aplicações Desktop e Web. Por conseguinte, será dada uma introdução ao assunto, em vista do suporte oferecido pelo Delphi Prism, que se integra ao IDE do Microsoft Visual Studio em forma de plug-in.

***Para que serve:***

Aplicações WPF proporcionam interfaces ricas que aumentam a experiência com o usuário, resultando em um produto final de grande apelo visual. Paralelamente, o WPF ainda oferece um modelo de programação conciso, onde há uma clara divisão entre a parte gráfica da aplicação e sua lógica de negócios. Através do Delphi Prism, a comunidade Delphi está apta a desenvolver este tipo de aplicativo.

***Em que situação o tema é útil:***

O Windows Presentation Foundation foi lançado na versão 3.0 do .NET Framework da Microsoft, e faz uso de uma linguagem de marcação própria, denominada XAML (Extensible Application Markup Language). O WPF, assim como é popularmente conhecido, tem como principal objetivo proporcionar, através de seus recursos, interfaces mais ricas de usuário, a fim de aprimorar a experiência deste. Além disso, uma aplicação WPF pode ser facilmente estabelecida tanto em ambiente Desktop quanto Web.

***Introdução ao WPF:***

O presente artigo apresentará um contexto introdutório para o desenvolvimento com a tecnologia WPF (Windows Presentation Foundation), disponível para a comunidade Delphi através do Delphi Prism. Para isto, serão expostos alguns pontos que irão levar o leitor a compreender onde poderá ser empregado este tipo de aplicação em seu contexto pessoal e profissional. Durante a passagem dos conceitos gerais da tecnologia, serão expostos alguns exemplos práticos simples, de cunho totalmente didático, porém não menos importantes para o início de aprendizagem. Dessa forma, em artigos futuros, este trabalho poderá ser usado como referência e ponto de partida.

# Delphi Prism XE

Fabrício Hissao Kawata

O WPF é muito associado ao Windows Forms, que era a API predominante até a versão 2.0 do .NET Framework para desenvolvimento Desktop, sendo considerado seu sucessor. A diferença entre as duas tecnologias se dá primordialmente na questão gráfica. Enquanto o Windows Forms faz uso das funções gráficas tradicionais da API do Windows, o WPF é voltado para o DirectX, cuja API é focada basicamente em jogos e recursos multimídia, o que significa aproveitar toda a aceleração de hardware disponível.

Isto justifica a vantagem que o Windows Presentation Foundation leva, em termos gráficos, sobre o Windows Forms. Já como semelhança, ambas as tecnologias apresentam uma generosa biblioteca de componentes, além de trabalhar com conceitos já conhecidos como code-behind e data binding.

A curva de aprendizagem do WPF é significantemente maior que o do tradicional Windows Forms, entretanto, este custo é compensado pelos benefícios proporcionados por algumas das próprias características do Windows Presentation Foundation, como será visto a seguir.

## Linguagem declarativa de interface de usuário

O WPF faz uso de uma linguagem de marcação própria, criada pela Microsoft, baseada em XML, para a construção de interfaces gráficas, denominada XAML, um acrônimo de Extensible Application Markup Language. Em termos práticos, a linguagem pode ser comparada ao uso do HTML no que tange à forma de declarar os elementos visuais no código. Uma das vantagens oferecidas por este tipo de recurso é a possibilidade de dividir o trabalho gráfico da codificação em si. Numa situação meramente ilustrativa, o XAML ficaria a cargo do profissional de designer da empresa, onde este poderia usar todo o seu poder de elaboração gráfica para as telas do sistema que o usuário estará interagindo. Ao mesmo tempo, o seu code-behind, ou seja, o código de negócio da aplicação ficaria sob domínio exclusivo da equipe de desenvolvimento, evitando dessa forma a interferência cruzada nos trabalhos de cada grupo profissional.

### Nota do DevMan

Microsoft DirectX é um conjunto de APIs voltadas para a programação de jogos, que padroniza a comunicação entre hardware e software, fornecendo informações para que ambos utilizem da melhor forma seus recursos. É exclusiva para o sistema operacional Microsoft Windows.

O XAML pode ser compilado ou interpretado. A primeira situação ocorre nos casos onde se tem código (C#, Oxygene etc.) incorporado ao XAML, que é o que ocorre quando se trabalha com uma aplicação WPF dentro do Delphi Prism. Neste caso, todo o XAML envolvido no projeto é compilado, resultando em um único executável. Entretanto, há casos em que você pode escrever um XAML simples, utilizando somente linguagem declarativa, num editor de texto comum e que será interpretado, por exemplo, por um navegador Web.

## Layout eficaz e gráficos vetoriais

A sistemática de layout apresentada pelo WPF fornece um meio inteligente em termos de organização de seus componentes gráficos. Com uma gama de opções de definição de layout, o ajuste e redimensionamento dos inúmeros componentes visuais na tela de um aplicativo se dão adequadamente e automaticamente.

Além disso, o WPF faz uso de gráficos vetoriais para construção de interfaces de usuário, que são imagens baseadas em vetores matemáticos. Este tipo tem como uma de suas principais características ser mais leve, se comparado às tradicionais imagens raster, baseadas em pixels, ou seja, ocupam menos espaço em disco. Outra vantagem é que gráficos baseados em vetores não perdem qualidade visual quando ocorre sua ampliação, deixando-os totalmente escaláveis. Isto já não ocorre em gráficos raster.

## Composições, customizações e modelos

Através do WPF é possível se montar composições de controles das mais variadas formas e combinações, visto a total abertura em se embutir controles dentro de controles. Um exemplo é a possibilidade de se utilizar um componente ComboBox, cujos itens são botões (Buttons).

Ainda sobre o tema, pode-se dizer que o termo reutilização é uma constante em WPF, e diz respeito à criação de elementos que possam ser reutilizados nas mais diversas interfaces de usuários que venham a ser construídas. Isso se dá de duas formas básicas, através do uso de templates (modelos): modelos de controle e modelos de dados. Em outros meios, você verá estes itens sendo tratados como Control Templates e Data Templates. O primeiro modelo se refere à definição da forma visual de um controle, enquanto o segundo trata da forma de renderização dos dados da aplicação.

O WPF ainda permite uma fácil customização da aparência da aplicação, através de recursos como Styles, Skins e Themes. Por exemplo, os estilos (Styles), são folhas de estilo responsáveis

por englobar características visuais de determinados elementos, da mesma forma como ocorre em HTML. Todavia, em WPF, é possível a declaração também de elementos não-visuais, além de poderem ser combinados com os templates (modelos), anteriormente citados.

## Data Binding

Data Binding (ou ligação de dados) é uma técnica de programação que interliga duas fontes de informações, mantendo-as em sincronia, sejam elas de diferentes tipos ou não. Por se tratar de um conceito genérico, pode ser desmembrado em outros específicos, tais como: UI Data Binding, XML Data Binding etc.

O WPF apresenta o Data Binding como um de seus poderosos recursos, possibilitando a ligação dos dados de fontes distintas, sem a necessidade de código. Um exemplo clássico é poder ter um objeto, cujas propriedades estão ligadas a controles visuais de uma interface gráfica, onde esta interligação é de maneira unidirecional (apenas um dos lados sofre alterações), ou bidirecional (uma mudança em um dos lados é refletida no outro).

O Data Binding no WPF é declarativo, o que significa que você não precisa escrever código para se conseguir essa ligação de dados, ao invés disso, somente é necessário descrever como se dará essa relação. Por exemplo, ao fazer o "binding" entre uma coleção de strings e um controle ListBox, você não precisará escrever todo o código que irá percorrer toda a coleção, adicionando cada item no ListBox. Somente é preciso descrever a relação que ambos terão.

## Otimização de processamento

O WPF, como já mencionado, possui uma relação com o DirectX, sendo construído em Direct3D. Este pequeno detalhe resulta em um melhor desempenho no processamento de seu conteúdo (gráficos e textos), sejam eles 2D ou 3D. Isso é conseguido devido ao WPF converter automaticamente este conteúdo em objetos do próprio Direct3D, para que sejam processados pelo hardware e alcançando uma melhor aceleração deste.

## WPF, Silverlight e Blend

Quando se fala em WPF, dificilmente é deixado de citar o seu co-irmão Silverlight, que é uma plataforma para o desenvolvimento RIA (Rich Internet Applications) e que foi inicialmente apresentada como WPF/E (Windows Presentation Foundation Everywhere). Dessa forma, a Microsoft utilizou a premissa de que o foco principal do WPF eram as aplicações Desktop enquanto que ao Silverlight cabia o ambiente Web, com foco em recursos multimídia e compatibilidade multiplataforma (cross-platform). Tornou-se um forte concorrente ao Adobe Flash, justamente devido a estes recursos.

O Silverlight, assim como o WPF, também é baseado na linguagem de marcação XAML, portanto muito do conhecimento adquirido em um, pode ser eficazmente aproveitado no outro. Além do XAML, o Silverlight possui um forte trabalho com JavaScript, e este casamento é o que basicamente forma suas aplicações.

Desde o seu lançamento, em meados de 2007, o Silverlight também é muito utilizado no desenvolvimento de LOB (Line-Of-Business) Applications, fugindo um pouco de sua intenção inicial. No entanto, essa fluência se deu de forma natural pela comunidade que o utiliza, visto sua praticidade em proporcionar a uma aplicação que roda em um navegador Web, as mesmas praticidades oferecidas por aplicações Desktop. Esta mais recente tendência o habilitou como sendo uma alternativa ao Adobe Flex, que possui os mesmos propósitos.

Para mais informações, na seção links você encontra o endereço da página oficial do Silverlight que, até o fechamento deste artigo, se encontra em sua versão de número 4.

O Microsoft Expression Blend é uma ferramenta focada em design e parte integrante de uma família de produtos denominada Expression, da Microsoft, voltada para os profissionais deste ramo. É totalmente compatível com o Visual Studio, sendo muito utilizada para a criação de interfaces gráficas ricas e animações, para aplicações WPF e Silverlight, gerando todo o código XAML necessário para estes fins. O Blend, como assim é tratado, pode facilmente exportar seus arquivos para o Visual Studio, auxiliando na colaboração entre as equipes de desenvolvimento e design. Para mais detalhes da ferramenta, o link de seu site oficial se encontra no final do artigo.

## Hello World

Partindo essencialmente para a utilização do Delphi Prism, as aplicações WPF possuem um tipo específico de projeto na IDE, conforme pode ser visto na **Figura 2**. Este é o ponto de partida para que se obtenha toda uma estrutura para a construção deste tipo de aplicativo.

Para se compreender o funcionamento da linguagem declarativa XAML, nada melhor que uma demonstração de um caso prático, no melhor estilo "Hello World", como o da **Figura 3**.

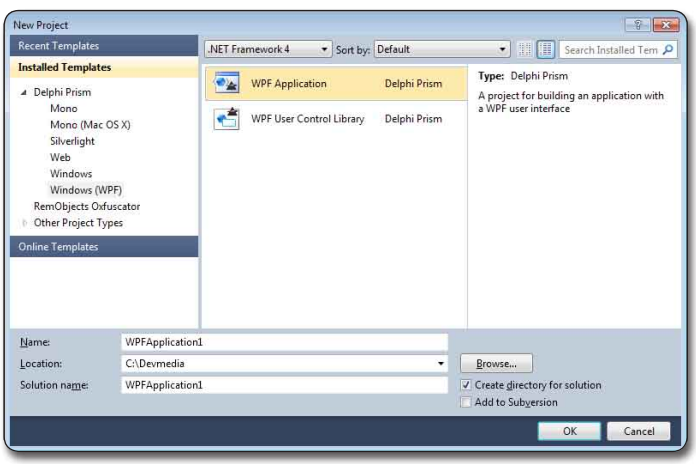



**Figura 2.** Tipo de projeto WPF Application

Ao mesmo tempo, na **Listagem 1** você pode conferir o código completo desta janela, que simplesmente escreve o texto "Alô Mundo!", num componente Label padrão.

**Figura 3.** Primeiro exemplo básico - Hello World

Antes de qualquer comentário, considere um elemento XAML como uma instância de um objeto, cujos atributos são as propriedades deste objeto. Neste trecho de código, percebe-se claramente a proximidade do XAML com o XML tradicional. Na linha 2 temos a declaração inicial do elemento raiz da estrutura que, neste caso é uma instância da classe System.Windows.Window, que é uma janela que englobará todos os demais componentes em tela.

Ainda no elemento raiz, por padrão, são definidos dois namespaces adotados convencionalmente pela Microsoft, conforme pode ser visto nas linhas 3 e 4. Não bastasse a tag Window, além dela, é necessário termos um container que conterá os componentes visuais que, neste exemplo, é definido pela tag Grid, a partir da linha 6. Logo, dentro dele está declarado seu elemento filho que é uma instância de System.Windows.Controls.Label, utilizado para exibir o texto de "Alô Mundo!" através de sua propriedade Content.

**Listagem 1.** Código XAML para um simples "Hello World"

```
01 <?xml version='1.0' encoding='utf-8' ?>
02 <Window x:Class="WPFApplication1.Window1"
03     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
04     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
05     Title="Hello World" Height="300" Width="300">
06     <Grid>
07         <Label Content="Alô Mundo!"
08         VerticalAlignment="Center"
09         HorizontalAlignment="Center" />
10     </Grid>
11 </Window>
```

## Arquivos do projeto

Olhando para o projeto criado (**Figura 4**), mais especificamente sobre sua estrutura, o IDE criou automaticamente dois arquivos com extensão .xaml: App.xaml e Window1.xaml. Junto de cada um deles, foi criado também o seu arquivo de code-behind, de extensão .pas, como os do Delphi tradicional Win32. Automaticamente, o IDE nos exibe para trabalho o visual do arquivo Window1.xaml. Mas o que ocorre com App.xaml? Essa e outras explicações serão apresentadas a seguir.

## App.xaml

Sempre que tiver dúvida sobre o que vem a se este arquivo App.xaml, criado junto de seu projeto WPF, interprete-o como sendo um item não visual, que representa a aplicação em si. Entre suas funções principais estão a de armazenamento de recursos (resources) a serem usados na aplicação e a definição de qual será o ponto de partida do aplicativo, ou seja, a janela inicial do sistema quando este for executado.

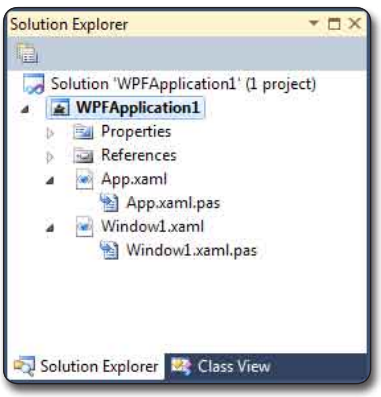



**Figura 4.** Estrutura do projeto na Solution Explorer

Na **Listagem 2** é mostrado o código do arquivo. Ilustrando o que foi dito anteriormente, o código de App.xaml representa a aplicação em si, conforme pode ser observado a partir da linha 2, onde temos a declaração de uma instância de System.Windows.Application. Na linha 5, a propriedade StartupUri é quem define o arquivo XAML que será exibido primeiro ao se iniciar a aplicação WPF em questão. Finalmente, a partir da linha 6 é declarada a propriedade Resources da instância Application, onde deverão ser definidos os recursos que por ventura venham ser utilizados pela aplicação.

**Listagem 2.** Código de App.xaml

```
01 <?xml version='1.0' encoding='utf-8' ?>
02 <Application x:Class="WPFApplication1.App"
03     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
04     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
05     StartupUri="Window1.xaml">
06     <Application.Resources>
07
08     </Application.Resources>
09 </Application>
```

## Window1.xaml

Uma vez apresentados os conceitos em torno de App.xaml, o outro arquivo, Window1.xaml, como já foi mencionado anteriormente, é a parte visual apresentada pela IDE para trabalho, logo ao se criar uma aplicação do tipo WPF. Resumindo, Window1.xaml é a janela principal do aplicativo, tanto que em seu código declarativo estão presentes as marcações de todos os elementos visuais representativos da aplicação em tela, bem como o comportamento que estes venham a ter.

Na **Listagem 3** temos um exemplo da codificação de uma janela principal de uma aplicação. No código desta listagem você percebe a mesma sistemática apresentada na **Listagem 1** visto a declaração das instâncias dos diversos objetos, como Buttons, TextBox e Labels. O resultado visual deste código pode ser visto na **Figura 5**.

**Figura 5.** Resultado visual da **Listagem 3**

## Layout

Layout é um item fundamental a se compreender, no que diz respeito ao início de aprendizado do WPF, uma vez que este tema estará presente em toda e qualquer aplicação que venha a ser desenvolvida com esta tecnologia.

Dito isto, depois de apresentados alguns exemplos de sua capacidade visual, veremos que o Windows Presentation Foundation oferece alguns recursos voltados para a organização do layout de uma aplicação. Na verdade são cinco opções, todas pertencentes ao namespace System.Windows.Controls, que visam facilitar o arranjo ideal dos elementos visuais em tela. São eles: Canvas, StackPanel, WrapPanel, DockPanel e Grid.

## Canvas

Comecemos pela opção mais básica de layout em WPF, que é o Canvas, um painel simples, que suporta o posicionamento dos elementos visuais através de coordenadas explícitas. Essa sua característica o apresenta como um painel primitivo, não sendo talvez a opção mais recomendada para a criação de interfaces de usuários mais elaboradas e flexíveis, mas sim para o posicionamento de desenhos e formas primitivas. Canvas também é o mais leve dentre todas as opções de layout. Na **Listagem 4** é apresentado um trecho de código que demonstra o uso de Canvas, onde o resultado final pode ser visto na **Figura 6**. Uma observação a se fazer é que a base deste exemplo, ou seja, os três Buttons dispostos, será utilizada para a demonstração das demais opções. Dessa forma, você facilmente perceberá a diferença entre cada opção de layout a ser usada em aplicações WPF.

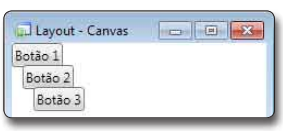



**Figura 6.** Posicionamento dos elementos visuais utilizando Canvas

## StackPanel

Da mesma forma que dizemos que Canvas é a opção mais básica e talvez não tão usual, podemos dizer que StackPanel é um dos mais populares e usuais. Sua característica principal é sua maneira de dispor seus elementos visuais filhos, através do empilhamento em sequência destes.

Novamente, para ajudar a fixar conceitos, é apresentado na **Listagem 5** o trecho de um código XAML que utiliza StackPanel, e na **Figura 7** seu resultado final.

## WrapPanel

O WrapPanel é muito parecido com a opção anterior StackPanel, ou seja, também organiza seus elementos filhos através do empilhamento destes. A diferença fica por conta nos casos específicos onde não há espaço suficiente para que os elementos sejam dispostos em uma única pilha. Nestas situações, o WrapPanel simplesmente move os elementos restantes para linhas ou colunas adicionais. Por este motivo, o WrapPanel é mais comumente usado dentro de outros controles e não como o container principal de uma janela. Para um melhor entendimento, veja na **Listagem 6** a demonstração de uso do WrapPanel e o que acontece quando não há espaço físico para se empilhar todos os elementos declarados em uma única pilha (**Figura 8**).

**Listagem 3.** Exemplo de código XAML de uma janela de sistema

```
01 <?xml version='1.0' encoding='utf-8' ?>
02 <Window x:Class="WPFApplication1.Window1"
03   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
04   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
05   Title="Autenticação de Usuário" Height="300" Width="300">
06   <Grid>
07     <Label Name="lblUserID" Content="User ID:"
08         Height="28"
09         Margin="27,86,0,0"
10         VerticalAlignment="Top" HorizontalAlignment="Left" />
11     <TextBox Name="txtUserID"
12          Height="23" Width="120"
13          VerticalAlignment="Top" HorizontalAlignment="Left"
14          Margin="96,86,0,0" />
15     <Label Name="lblPassword" Content="Password:"
16         Height="28"
17         Margin="27,113,0,0"
18         VerticalAlignment="Top" HorizontalAlignment="Left" />
19     <TextBox Name="txtPassword"
20          Height="23" Width="120"
21          VerticalAlignment="Top" HorizontalAlignment="Left"
22          Margin="96,115,0,0" />
23     <Button Name="btnLogin" Content="Login"
24         Height="23" Width="75"
25         VerticalAlignment="Top" HorizontalAlignment="Left"
26         Margin="52,147,0,0"
27         Click="btnLogin_Click" />
28     <Button Name="btnCancel" Content="Cancel"
29         Height="23" Width="75"
30         VerticalAlignment="Top" HorizontalAlignment="Left"
31         Margin="141,147,0,0"
32         Click="btnCancel_Click" />
33   </Grid>
34 </Window>
```

**Listagem 4.** Trecho de código utilizando Canvas

```
<Canvas>
  <Button Canvas.Left="0"
      Canvas.Top="0">Botão 1</Button>
  <Button Canvas.Left="10"
      Canvas.Top="20">Botão 2</Button>
  <Button Canvas.Left="20"
      Canvas.Top="40">Botão 3</Button>
</Canvas>
```

**Listagem 5.** Trecho de código utilizando StackPanel

```
<StackPanel>
  <Button Content="Botão 1" />
  <Button Content="Botão 2" />
  <Button Content="Botão 3" />
</StackPanel>
```

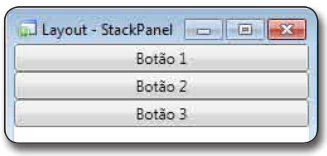



**Figura 7.** Posicionamento dos elementos visuais utilizando StackPanel

**Figura 8.** Posicionamento dos elementos visuais utilizando WrapPanel

## DockPanel

DockPanel é um tipo de painel especial que permite que seus elementos filhos sejam devidamente encaixados em um de seus quatro lados, possibilitando ainda o preenchimento de toda sua altura ou largura. Ele ainda permite que um elemento preencha toda sua área restante, não utilizada pelos outros elementos já encaixados. Para elucidar sua utilização, tome como base o trecho de código da **Listagem 7**, que resulta no layout mostrado na **Figura 9**.

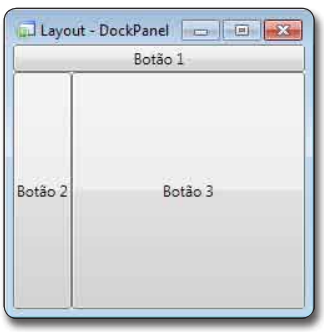



**Figura 9.** Posicionamento dos elementos visuais utilizando DockPanel

## Grid

Completando as opções de layout no WPF, temos o Grid que, como você já deve ter observado, é apresentado como layout padrão ao se criar uma aplicação do tipo WPF Application no Delphi Prism. Essa escolha da IDE se deve ao fato de ele ser o painel mais versátil dentre os demais e talvez o mais usado como container principal. Sua forma de organizar seus elementos filhos se dá através de linhas e colunas, muito similar a uma tabela em HTML. Na **Listagem 8** você tem um trecho XAML, onde se utiliza o Grid como layout, tendo os três botões de nosso exemplo como seus elementos filhos. O resultado final deste código é visto na **Figura 10**.

No código apresentado, a estrutura de Grid foi definida com três linhas e três colunas, indicadas em suas propriedades RowDefinitions e ColumnDefinitions, conforme apresentado entre as linhas de 2 a 6 e de 8 a 12 na listagem. Feito isto, cada elemento filho pode ser devidamente posicionado em uma posição específica da tabela, conforme mostra as linhas 14 a 16.

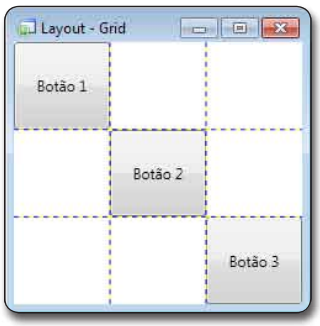



**Figura 10.** Posicionamento dos elementos visuais utilizando Grid

## Eventos simples

Como uma introdução a eventos no WPF, ainda na **Listagem 3**, dentre as propriedades do botão btnLogin, há uma apresentada da seguinte forma: Click="btnLogin_Click". Este trecho está indicando que o Click deste botão está vinculado a um evento denominado btnLogin_Click, que deverá estar declarado no arquivo de code-behind (.pas) associado ao arquivo .xaml em questão. De forma prática, voltando um pouco mais no artigo, vamos utilizar o primeiro exemplo, Hello World, para demonstrar a utilização de um evento declarado na marcação XAML.

Da forma que está, o código da **Listagem 1** resulta em uma interface que somente mostra o texto com os dizeres "Alô Mundo!" estaticamente. Contudo, poderíamos acrescentar um simples botão em seu escopo que, ao ser clicado, exibiria uma caixa de mensagem com os mesmos dizeres. Para que isso seja possível, um simples duplo-clique neste botão irá declarar e criar

**Listagem 6.** Trecho de código utilizando StackPanel

```
<StackPanel>
  <Button Content="Botão 1" />
  <Button Content="Botão 2" />
  <Button Content="Botão 3" />
</StackPanel>
```

**Listagem 7.** Trecho de código utilizando DockPanel

```
<DockPanel>
  <Button DockPanel.Dock="Top" Content="Botão 1" />
  <Button DockPanel.Dock="Left" Content="Botão 2" />
  <Button Content="Botão 3" />
</DockPanel>
```

**Listagem 8.** Trecho de código utilizando Grid

```
01 <Grid ShowGridLines="True">
02     <Grid.RowDefinitions>
03       <RowDefinition />
04       <RowDefinition />
05       <RowDefinition />
06     </Grid.RowDefinitions>
07
08     <Grid.ColumnDefinitions>
09       <ColumnDefinition/>
10       <ColumnDefinition/>
11       <ColumnDefinition/>
12     </Grid.ColumnDefinitions>
13
14     <Button Grid.Row="0" Grid.Column="0"  Content="Botão 1" />
15     <Button Grid.Row="1" Grid.Column="1"  Content="Botão 2" />
15     <Button Grid.Row="2" Grid.Column="2"  Content="Botão 3" />
16 </Grid>
```

automaticamente seu método, assim como acontece no desenvolvimento Delphi tradicional. Como vimos, a declaração se dá no XAML, onde teríamos algo como Click=”button1_Click”. Já o método criado se encontra em seu code-behind:

```
method Window3.button1_Click(sender: System.Object; e: System.Windows.
RoutedEventArgs);
begin
  MessageBox.Show(“Alô Mundo!”);
end;
```

## Conclusão

Neste artigo vimos uma explicação introdutória do que vem a ser o WPF e como sua parceria com o Delphi Prism permite a criação de interfaces ricas e sofisticadas de usuário em ambiente Windows. Indo um pouco além, o WPF oferece ainda suporte a animações, multimídias (áudio e vídeo), desenhos 3D, tipografia e muitos outros tópicos não abordados aqui, mas que poderão aparecer em artigos futuros. Contudo, se houve o interesse neste tipo de desenvolvimento, não perca tempo e se aprofunde no assunto que é bem extenso.

**Fabrício Hissao Kawata**
*fabricio.kawata@bol.com.br*
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.

**Delphi Prism XE - Página de apresentação do produto**
*http://www.embarcadero.com/products/delphi-prism*

**Delphi Prism XE – Página para download**
*https://downloads.embarcadero.com/free/delphi_prism*

**WPF - Página oficial**
*http://windowsclient.net*

**Silverlight - Página oficial**
*http://www.silverlight.net*

**Microsoft Expression Blend - Página oficial**
*http://www.microsoft.com/expression/products/Blend_Overview.aspx*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Bancos de Dados Gratuitos

## Componentes de conexão

Vitor Luiz Rubio

Em qualquer livro sobre bancos de dados é possível ler uma introdução onde se fala sobre a evolução dos mesmos, passando de simples arquivos a bancos hierárquicos, como estruturas de diretórios, e daí para bancos relacionais.

Bancos de dados relacionais são baseados em tabelas e os relacionamentos entre elas. Estas tabelas são sempre bidimensionais, onde no eixo "x" têm-se as colunas, e no eixo "y" têm-se os registros. Um registro é formado por dados, ou valores, para as respectivas colunas. Então se uma tabela tem três colunas, como "Nome", "Telefone" e "e-mail" um registro teria três dados, ou valores, um para cada coluna. Por exemplo (Vitor, 11 12345678, email@servidor.com.br) seria um registro, enquanto que (José, 11 98765432, jose@umservidor.com.br) seria outro registro.

Os bancos de dados relacionais provêm uma maneira única de qualquer aplicação acessar qualquer banco de dados. Fazem isso através de um padrão de armazenamento próprio e de um SGBD. O SGBD, ou sistema gerenciador de banco de dados, cria uma interface entre o programa e o banco de dados em si e é ele quem grava ou lê os dados do banco. O programa não precisa conhecer as complexas estruturas de dados e índices usados no arquivo de banco de dados para acessá-los. Basta saber como se comunicar com o SGBD que este fará todo o processo. O SGBD assume a responsabilidade de ler, gravar, garantir ou revogar acesso, indexar e manter o banco de dados. Para que o programa conheça e saiba se comunicar com o SGBD é necessário ainda outra camada, que são os drivers de bancos de dados, também chamados de bibliotecas clientes. A comunicação é feita através da Structured Query Language ou SQL.

O SQL e o modelo relacional fornecem um padrão para que absolutamente qualquer conjunto de dados, de qualquer tipo, possa ser representado numa estrutura de tabelas, colunas, registros e relacionamentos. Na linguagem formal os registros também são chamados de tuplas, as tabelas de entidades (ou relações) e as colunas de atributos.

Qualquer banco de dados relacional pode ser facilmente representado graficamente em um diagrama chamado diagrama de entidade-relacionamento (DER) ou modelo de entidade-relacionamento (MER). Muitos desenvolvedores consideram esse diagrama um dos melhores instrumentos para representar o negócio de uma empresa e apresentar para os stakeholders (clientes e outros interessados).

Muitos desenvolvedores partem de um diagrama desse tipo para desenvolver todo o banco de dados e o sistema, e criar um sistema totalmente orientado aos dados é um método preferido por muitos desenvolvedores.


## Resumo DevMan

***De que trata o artigo:***

Este artigo trata sobre bancos de dados gratuitos, proprietários ou open-source, e das maneiras disponíveis no mercado para se conectar a eles. Serão apresentados os bancos de dados, o passo a passo para sua instalação e os componentes para conexão.

***Para que serve:***

Bancos de dados servem para armazenar, organizar, indexar e buscar informações. Os bancos de dados mais utilizados são os relacionais. Todos os sistemas de informação necessitam de bancos de dados, exceto sistemas que, por questões de performance, fazem processamento de certos arquivos em lote. Bancos de dados gratuitos podem eliminar alguns custos tanto do desenvolvedor como dos clientes.

***Em que situação o tema é útil:***

Sempre que se começar um projeto novo, do zero, o analista ou arquiteto passará pelas fases de análise, concepção, desenho e assim por diante. Em uma dessas etapas o modelo de dados conceitual, ou de negócios, será desenhado, e a partir daí o modelo de dados lógico. Desse ponto em diante, tanto nas fases de desenvolvimento como nas fases de teste e homologação, os desenvolvedores necessitarão saber como se instala e como se conecta a um sistema gerenciador de bancos de dados.

***Bancos de Dados Gratuitos :***

É conhecido e indiscutível o fato de que bancos de dados relacionais têm uma série de vantagens sobre arquivos de dados ou de texto, que é como os dados eram armazenados antigamente. Este artigo visa apresentar as principais opções de bancos de dados gratuitos e como se conectar a eles. Formas diferentes de conexão a um mesmo banco de dados são conhecimentos necessários, pois empresas diferentes acessam os bancos de dados de maneiras diferentes, levando em consideração ao que já foi desenvolvido, sistemas legados e frameworks adquiridos. Além disso, em sistemas onde o banco de dados é proprietário e já foi definido e adquirido, usar versões gratuitas ou Express desses bancos pode diminuir os custos de teste e homologação, além de ajudar no treinamento e capacitação do desenvolvedor.


Para quem programa orientado a objetos outra camada (ou várias) é necessária: a camada de ORM, ou mapeamento objeto-relacional. Essa camada seria responsável por mapear, ou transformar, os objetos de negócio em um conjunto de dados de tal forma

que possa ser representado por tabelas e campos e armazenados em um banco de dados relacional.

Não é objetivo desse artigo abordar toda a teoria de bancos de dados, afinal existem livros para isso. Esse artigo tem por objetivo apresentar os principais bancos de dados gratuitos e mostrar como se conectar a eles. Antes de tudo é importante definir a diferença entre software gratuito, software open-source e software livre.

Software gratuito, como o próprio nome diz, não tem custo para quem o usa. Isso não quer dizer que ele possa ser usado sem restrições. A maioria dos bancos de dados "Express" tem restrições quanto ao número máximo de processadores, de usuários, de memória ou até mesmo se o uso é pessoal ou comercial.

Diferentemente deste tipo os bancos de dados open-source podem ser usados sem restrições e ainda o sistema gerenciador de banco de dados (SGBD) tem o código-fonte aberto, podendo-se visualizar, alterar e recompilar esse código fonte.

Software livre é um subconjunto de software open-source cuja licença é compatível com a licença GPL. Isso é uma garantia de que tanto o código fonte como outras partes do software, como bibliotecas, logotipos e etc. também são livres, e que softwares derivados deste também são livres. Quem dita as regras da GPL e diz se uma licença é compatível ou não com a GPL é a Free Software Foundation.

## Drivers

Para que um programa feito em uma linguagem qualquer se comunique com um sistema gerenciador de banco de dados qualquer é necessário um driver. Driver nesse contexto significa um programa ou DLL que é capaz estabelecer uma comunicação entre o programa desenvolvido e o SGBD.

Os SGBDs abrem portas TCP, UDP ou ambas para que os programas possam se comunicar com eles. Todas as transferências de dados ocorrem por essas portas. O que o programa envia para o SGBD são comandos SQL de inserção, atualização ou de consulta, e o que o SGBD responde pode ser a quantidade de registros alterados ou os registros em si.

Os drivers servem para que o programador não tenha que saber detalhes específicos de tcp/ip, muito menos como cada banco de dados específico trabalha essas "conversas" de dados.

**Nota**

A escolha de um banco de dados é uma parte muito importante de um projeto de software, assim como conhecer as características, vantagens e desvantagens de cada um. Cada banco de dados tem limites de carga, performance e ambiente, porém se o seu projeto não pretende ou não prevê tocar em nenhum desses limites, ou se é apenas um protótipo ou um teste, como no caso desse artigo, não é necessário se importar com o banco de dados. Qualquer um servirá.

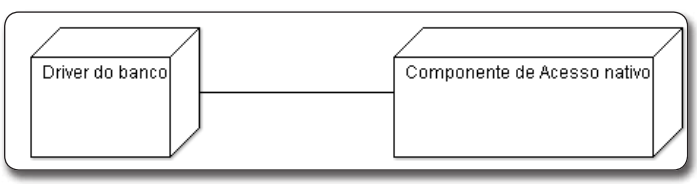



**Figura 1.** Componentes de acesso direto a um determinado banco de dados

Cada empresa fabricante de SGBD fornece uma API ou um conjunto de APIs, que podem ser dlls ou outros artefatos que implementam a comunicação básica com esse banco de dados. E cada fabricante de compiladores ou IDEs produz outra biblioteca, ou conjunto de bibliotecas, responsável por se comunicar com essas dlls.

Nada impede, como pode ser observado em alguns casos, que um fabricante de compiladores, IDEs ou bibliotecas escreva um driver direto para um determinado SGBD que se comunique diretamente com ele via tcp/ip.

Isso quer dizer basicamente que qualquer linguagem de programação pode produzir programas que se conectem com qualquer banco de dados, contanto que exista um driver que faça essa comunicação. Esse driver pode até ser desenvolvido por terceiros e adquirido separadamente.

Há drivers que acessam diretamente um determinado tipo de SGBD (**Figura 1**), e são chamados nativos. Há drivers que simplesmente oferecem uma camada comum e padrão de acesso aos dados pelo programa, mas necessitam de outros drivers para acessar o banco de dados realmente. É o caso do OLEDB e ODBC. Com eles é possível acessar qualquer banco de dados da mesma maneira, como se existisse um único SGBD, mas é o OLEDB ou ODBC que realmente acessa o banco de dados usando para isso outros drivers (**Figura 2**).

Há mecanismos que fornecem uma interface padrão para que um programa possa acessar diferentes bancos de dados com seus drivers nativos. Para isso ele usa várias bibliotecas que implementam essa interface, cada biblioteca responsável pelo acesso a um banco de dados diferente. É o caso do DBExpress do Delphi, que se conecta com bibliotecas (DLLs) diferentes e drivers diferentes, mas provê para um programa feito em Delphi o mesmo mecanismo para todos eles.

No Delphi os mecanismos de acesso a dados estão encapsulados em componentes, mas isso não significa que seja obrigatória a existência de componentes para acessar o banco de dados. Algum fabricante de SGDB pode, se quiser, fornecer uma classe para o Delphi acessar seu banco de dados que não seja descendente de TComponent.

É interessante observar que a adição de uma camada a mais de acesso diminui a performance. O quanto a performance será diminuída depende de como foi implementada a camada de acesso. Por exemplo, o acesso a um banco de dados qualquer através de ODBC ou OLEDB é mais lento do que o acesso direto. Esse provider pode ser usado no ADO (dbGo) por exemplo, mas fornece duas camadas intermediárias entre o sistema e o driver do banco de dados (**Figura 3**).

As políticas de cache, controle de recursos e controle de cursores depende de cada driver, camada intermediária e componente de acesso. Os componentes dbExpress fornecem uma solução bem limpa nesse contexto pois delegam ao ClientDataSet a responsabilidade de cuidar de cache e cursores (**Figura 4**).

A **Figura 1** mostra como atuam os componentes de acesso direto e específico a um único banco de dados sem passar por camadas intermediárias. Exemplos de arquiteturas como a da **Figura 1** seriam as arquiteturas IBX, para acesso exclusivo a interbase/Firebird, assim como a IBO, também para interbase/Firebird,

ou os componentes próprios correspondentes das bibliotecas Unidac e Zeos.

Esses são os componentes que acessam nativamente um banco de dados, que fazem referência ao nome do banco de dados no próprio nome do componente.

A **Figura 2** indica como o acesso aos diversos bancos de dados são feitos através de drivers ODBC ou OLEDB. Se por exemplo o projeto acessar um banco de dados qualquer através de um driver ODBC e o componente de acesso BDE, o TDatabase se conectará com o ODBC e o ODBC que se conectará com o driver do banco de dados apropriado. Se o acesso for feito através dos componentes da paleta ADO (dbGo) então o acesso será feito via OLEDB, mas conceitualmente é a mesma coisa: O componente AdoConnection acessará na verdade um driver OLDB e esse driver terá a responsabilidade de acessar a biblioteca "client" do banco de dados utilizado.

Existe no Windows um provider (ou driver) OLEDB para acesso ODBC, isso permite que através de ADO sejam acessíveis qualquer base de dados acessível via ODBC, inclusive bases de dados DBF (dbase), arquivos texto, arquivos csv, excell, Access e outros formatos. A desvantagem é que essa forma de acesso impõe mais duas camadas intermediárias entre a aplicação e seus dados.

A **Figura 4** mostra o DBExpress (DBX). Ele fornece uma interface padrão para acesso via aplicações em Delphi, mas possui um driver (uma DLL) para cada banco que ele acessar. Cada uma dessas DLLs se comunica de maneira direta com o client do banco de dados em questão.

Muitos desenvolvedores experientes afirmam que o acesso a um banco de dados via DBExpress é um acesso nativo porque realmente a camada do DBExpress é muito "fina". Os drivers do DBExpress (DLLs chamadas dbx*.dll) são apenas meios de fazer com que o SQLConnection se comporte como se fosse um componente diferente para cada banco de dados. Essas dlls acessam diretamente o client do banco de dados deixando toda política de cache e cursores a cargo do ClientDataset. É uma leitura "pura" ou "bruta" do banco de dados. Por isso os SQLDatasets não são usados diretamente para navegação nos registros, pois são unidirecionais.

Falar em um único artigo sobre cada um desses bancos de dados e meios de acesso é uma tarefa impossível. Um único banco de dados seria assunto para um livro, por isso esse artigo tem a intenção apenas de introduzir os bancos de dados e a maneira de acessá-los. Talvez em artigos futuros sejam apresentados cada um dos bancos de dados mais profundamente, mostrando inclusive como desenvolver sistemas completos, com tabelas, views, triggers e stored procedures.

Ao longo desse texto muitas vezes a expressão banco de dados será usada referindo-se ao SGDB e não ao banco de dados em si, principalmente quando o assunto for métodos de conexão.

## Firebird

Firebird é um banco de dados relacional, muito conhecido pela comunidade Delphi, porém menos conhecido por programadores de outras linguagens, que fornece todas as características padrão ANSI SQL. Apresenta versões que rodam em Linux, Windows e várias versões de Unix, 32 ou 64 bits. É multiusuário, suportando conexões concorrentes, oferece alta performance e suporte a triggers e stored procedures. Sua história está relacionada à história do Interbase, outro banco de dados muito conhecido pela comunidade Delphi.

Quem se lembra da época do Interbase 6 lembrará que o Firebird é um fork (ramificação) do interbase, e que embora no começo tentava manter compatibilidade total com o seu "primo", ambos evoluíram em direções diferentes, conforme as pressões do mercado e da comunidade.

Atualmente a versão estável do Firebird é a 2.5 e para instalá-lo basta fazer o download da versão mais apropriada para seu ambiente e seguir as orientações do programa de instalação. Existem

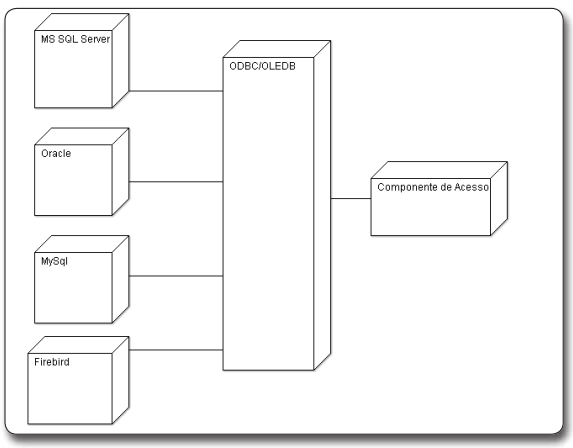



**Figura 2.** Diagrama de acesso a bancos de dados via ODBC ou OLEDB

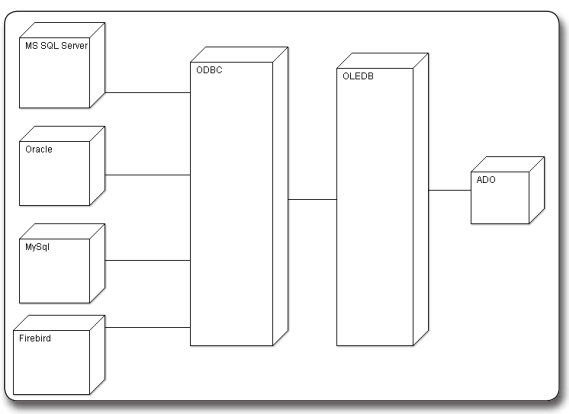



**Figura 3.** No caso do ado acessando ODBC via OLEDB

**Nota**

Os componentes IBX, assim como outros voltados a interbase são feitos para interbase e não para Firebird. Alguns permitem acesso a versões anteriores do Firebird através das bibliotecas de compatibilidade (gds32.dll), ou seja, somente os recursos compatíveis seriam usados e os exclusivos não. Alguns outros componentes com "IB" no nome, como IBObjects e UIB foram "otimizados" para trabalhar "também" com Firebird, independente da versão. Cabe ao programador consultar os manuais tanto do Firebird como dos componentes de acesso para verificar quais recursos estarão disponíveis. Entretanto recursos básicos como um select a uma tabela estarão disponíveis a todos.

também versões para serem embarcadas em instaladores de outros sistemas. Ou seja, é possível criar um instalador que instale o seu sistema mais o Firebird. Sua instalação segue o padrão Windows (Next, Next, Finish), porém a **Figura 5** apresenta alguns detalhes importantes de sua instalação.

As duas últimas opções do processo de instalação são interessantes: a penúltima pergunta se deve copiar a fbclient.dll para o diretório do sistema, marque essa opção para evitar ter que fazer isso mais tarde se no seu ambiente você não conseguir se conectar. A última opção pergunta se será necessário instalar a biblioteca gds32.dll para assegurar compatibilidade com programas feitos para acessar o Interbase. Depois disso é só clicar em instalar e depois em concluir, marcando a opção para iniciar o serviço do Firebird.

De acordo com o gnu.org a licença de uso do Firebird, a Interbase Public License 1.0 (IPL) é muito parecida com a Mozilla Public Licence 1.1 (MPL) tendo assim algumas restrições complexas. Por isso ela não é compatível com a licença GPL. Isso quer dizer que o Firebird é livre, mas não totalmente livre. Ele continua open-source e pode-se usar ele livre e gratuitamente sem problemas. Mas, analisando a licença, a gnu.org encontrou algumas restrições ou "não garantias" da liberdade do usuário que tornam a IPL uma licença não compatível com a GPL.

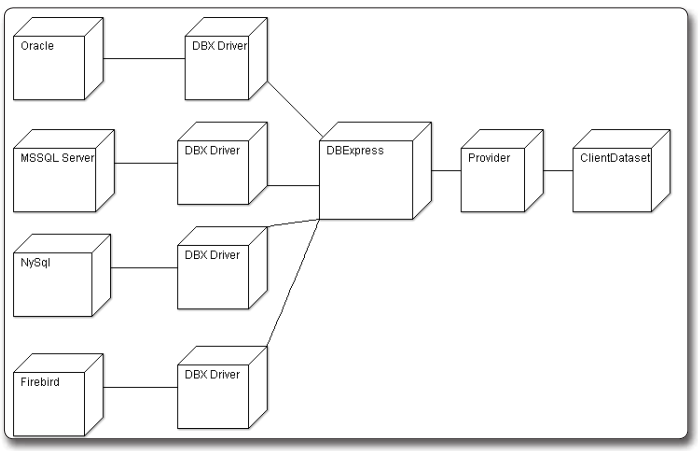



**Figura 4.** Ilustração da arquitetura do DBExpress

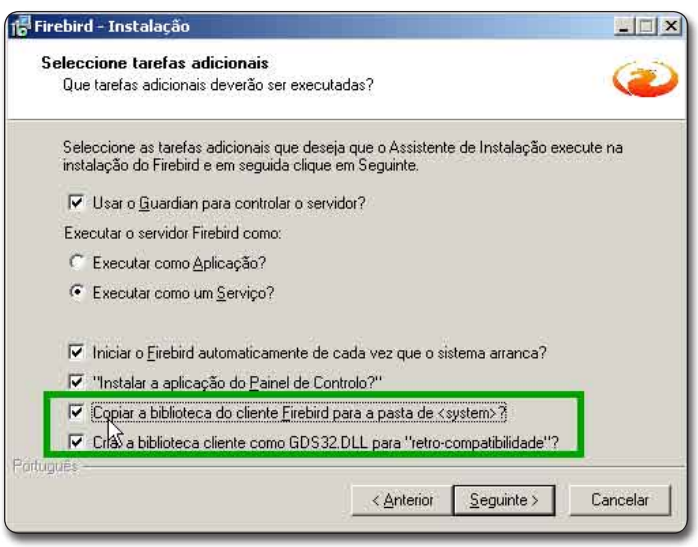



**Figura 5.** Penúltima etapa da instalação

**Nota**

O Firebird usa a porta 3050, por isso é importante que ela esteja liberada no firewall do ambiente no qual ele irá atuar.

A maneira mais fácil de se conectar ao Firebird é através dos componentes da seção dbExpress. Porém também é possível acessar o Firebird com IBExpress, BDE e ADO, ambos usando um driver ODBC, ou ainda via ADO usando um driver OLEDB.

O método de acesso a ser escolhido depende do ambiente que a empresa já tem desenvolvido, da versão do Delphi, da versão dos componentes de acesso e da versão do SGDB. Conhecer todos os métodos é importante para os casos de integração ou migração.

Para os testes de acesso ao banco de dados Firebird nesse artigo foi usado o banco de exemplo que vem junto com o SGDB, o employee.fdb, encontrado em C:\Arquivos de programas\Firebird\Firebird_2_5\examples\empbuild.

A versão do Delphi usada pode ser tanto a XE como a 2010. Na seção de links há o endereço de um provider OLEDB comercial (ibProvider). Uma versão trial pode ser baixada para ser testada por alguns dias, tempo suficiente para analisar os exemplos deste artigo. Além dos componentes e drivers que vêm junto com o Delphi, o Firebird também pode ser acessado pelos componentes UIB da biblioteca JEDI, pelos componentes Unidac, IBO e Zeos.

## MySQL

O slogan do site do MySQL é, numa tradução livre, "o banco de dados open source mais popular do mundo". Historicamente, o que realmente ajudou a popularizar o MySql foram as linguagens open-source voltadas principalmente para web, como php, Perl e python. O mySql é muito usado em sites na web, principalmente e-commerces e gerenciadores de conteúdo. Joomla e Wordpress são exemplos de CMS que armazenam os dados em MySQL.

O MySQL possui versões open source mas também possui versões enterprise, com um conjunto de ferramentas maior. A versão open source do MySql, chamada de community Server, pode ser baixada na url que encontra-se na seção de links desse artigo. A licença de uso do MySql é a licença GPL, ou seja, além de ser open source é completamente compatível com a filosofia da GNU Software Foundation.

Conectar-se ao MySql é possível através do driver para dbExpress. Drivers ODBC e OLEDB permitem a conexão usando BDE ou ADO.

Além do já citado uso nos ambientes web o MySQL se tornou popular pela facilidade de uso, não requer um DBA experiente para instalar ou usar. Outro fator decisivo é sua performance e confiabilidade. Ele pode rodar em uma série de ambientes, como Windows, Linux, Mac OS e vários "sabores" de Unix.

A instalação do MySql em ambiente Windows não tem nenhum segredo. Depois do processo de instalação será disparado o processo de assistente de configuração (Configuration Wizard). O configuration wizard é iniciado depois do processo de instalação. Essa configuração

permite a escolha da política de consumo de memória. Sempre que for possível dedicar todos os recursos exclusivamente ao banco de dados ele operará muito mais rápido. Também é possível determinar qual o ambiente de uso, se será em uma máquina de desenvolvimento, um servidor, ou um servidor dedicado. Escolhendo a opção Developer Machine, será otimizado o uso de memória e CPU para o ambiente de desenvolvimento. Pode-se também definir quantas conexões concorrentes serão permitidas e se o protocolo TCP/IP estará habilitado (permitindo conexões remotas). Durante essa configuração escolhe-se o suporte a caracteres estrangeiros/acentuados com UTF-8 (**Figura 6**). É importante que a porta 3306 esteja liberada no firewall do computador que será o servidor do MySql.

Na seção de links desse artigo há uma url para o download do MySql, outra para o download do Driver ODBC do MySql (que tem versões para 32 e 64 bits) e o download para o banco de dados de exemplo do MySql.

O MySql tem vários bancos de dados de exemplo, neste artigo foi usado um banco chamado employees, mas ele é muito diferente do employee.fdb do Firebird. Esse banco de dados vem no formato de arquivos com instrução SQL para criar as tabelas e arquivos maiores, ou dumps, com os comandos insert para popular as tabelas. Juntamente com tudo isso há um manual explicando como criar o banco.

## PostgreSQL

Postgre SQL é outro banco de dados relacional open source que pode ser baixado no site do fabricante. É possível encontrar versões Enterprise e Community. Numa tradução livre o postgre descreve a si mesmo como o "banco de dados open-source mais avançado do mundo".

Na leitura disponível na internet sobre o postgre é possível perceber uma certa pretensão de ser como um Oracle. De fato o postgre é um banco de dados muito bom, com muitos recursos avançados e que suporta uma carga de dados muito grande.

No processo de instalação deve-se dar atenção especial à etapa mostrada na **Figura 7**, que é a escolha de uma senha de superuser/administrador, que no caso do postgre se chama "postgres". Neste artigo foi escolhida a senha "123@abc" por conter letras, números e caracteres especiais, respeitando assim várias convenções comuns de políticas de senhas. A porta será a padrão, 5432.

Em certa posição do processo de instalação o instalador do postgre pergunta se deverá instalar software adicional. Existem muitos pacotes de software adicional para serem instalados, desde apache, passando por alguns CMSs como Drupal, por exemplo. Escolha apenas aqueles que forem necessários, como os drivers ODBC e OLEDB e o pgAdmin, para administrar o banco. O instalador do postgre já instalará automaticamente o driver OLEDB para acesso pelo ADO e um provider para acesso via .Net.

## Outros componentes de acesso

Antes de começar os testes é bom citar que os componentes que vêm junto com o Delphi não são os únicos possíveis de se usar. A empresa Devart produz uma excelente coleção de componentes para acessar vários bancos de dados, chamada Unidac. Há também

### Nota do DevMan

É necessário dizer que embora o MySql tenha licença GPL, ele hoje faz parte da Oracle. A Oracle é dona e principal mantenedora do MySql, sendo assim qualquer contribuição que um programador livre queira fazer ao MySql deve passar pelo crivo da Oracle. O fato dos dois bancos de dados serem rivais em uma camada do mercado onde a Oracle perdeu mercado para o MySql é visto como uma ameaça à continuidade do MySql. O lado bom de projetos open-source é que sempre que um projeto é ameaçado pela aquisição por uma grande empresa pode nascer um "fork" (bifurcação) ainda livre do projeto, como o MariaDB é um fork do MySql. Vale lembrar que o Firebird também é um fork do Interbase, mas mesmo a licença do Firebird não sendo compatível com a GPL, o Firebird não tem o rabo preso com nenhuma outra empresa de banco de dados e com nenhuma versão enterprise. Nesse sentido o Firebird é único: você adquire uma versão do Firebird e sabe que aquela versão é a melhor que existe. O Firebird já se distanciou bastante do Interbase e tem como filosofia ter sua própria evolução.

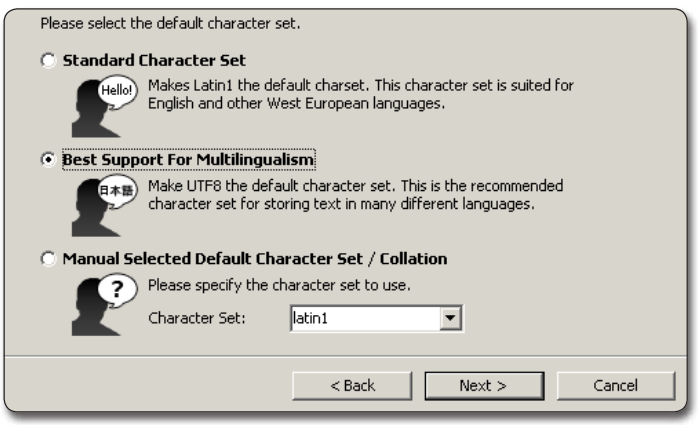



**Figura 6.** Suporte a caracteres estrangeiros

os componentes MyDac para acesso ao MySql, IbDac para acesso ao interbase / Firebird e Odac para acesso ao Oracle, para citar alguns exemplos. A Devart fabrica também drivers DBX alternativos aos que vêm junto com o Delphi. Vale a pena conhecê-los.

A aquisição desses componentes pode ser bastante interessante visto que eles procuram acessar um banco de dados da maneira mais otimizada possível, comunicando-se diretamente com o driver do SGBD e usando sabiamente os recursos como memória e cache.

Por último é bom que o desenvolvedor conheça o IBO (IB Objects), uma coleção de componentes de acesso ao Interbase e Firebird. A **Figura 8** mostra os providers instalados no processo de instalação do Unidac.

## Conectando-se ao Firebird

O exemplo mostra várias formas diferentes de se conectar ao Firebird:

1 – via DBExpress;
2 – Via IB Express;
3 – Via BDE com ODBC;
4 – Via ADO com ODBC;
5 – Via ADO com OLEDB;
6 – Via UniDac.

Os componentes UniDac não vêm junto com o Delphi e devem ser baixados do seu site, na seção de links. O objetivo de mostrar a cone-

xão de tantas maneiras diferentes é tanto fazer um benchmark para verificar qual método é mais rápido como apresentar para o leitor formas alternativas às tradicionais. Nesse teste apenas a consulta à tabela employee foi feita. Nenhum campo específico foi selecionado, ou seja, a consulta é feita com a query select * from CUSTOMER.

Para que os componentes de acesso possam ser testados separadamente é necessário um DBGrid e um DataSource para cada mecanismo de acesso, e um pagecontrol para acomodar todos os grids. Também é possível realizar o teste com um único DbGrid e um único DataSource mudando programaticamente o datasource do DataSet e abrindo-o, não sendo necessário dessa forma um pagecontrol com muitos dbGrids.

Para verificar o tempo que demora para abrir um dataset será usada a função GetTickCount antes e depois da abertura e a diferença mostrada em formato de segundos/milissegundos. Não esqueça de repetir esse teste com um banco de dados realmente grande.

Para se conectar via dbExpress é necessário adicionar um SQLConnection, um SQLdataset, um DatasetProvider e um ClientDataset. A propriedade Driver do SQLConnection deve ser Firebird, LibraryName deve ser dbxfb.dll (dependendo da versão do Delphi o nome da biblioteca para acesso do DBX ao Firebird pode mudar). A propriedade VendorLib sempre é a DLL client de acesso ao banco de dados. No caso do Firebird é a fbclient.dll, mas poderia ser a gds32.dll para casos de compatibilidade com versões mais antigas do Firebird ou Interbase 6.x.

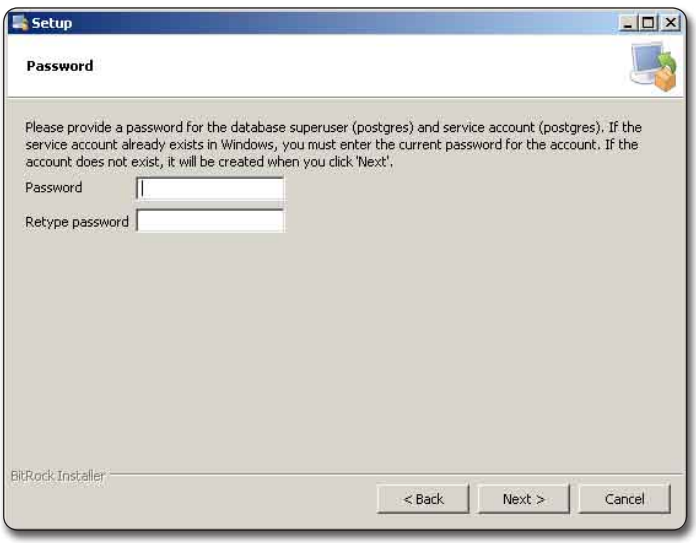



**Figura 7.** Escolha a senha do postgres

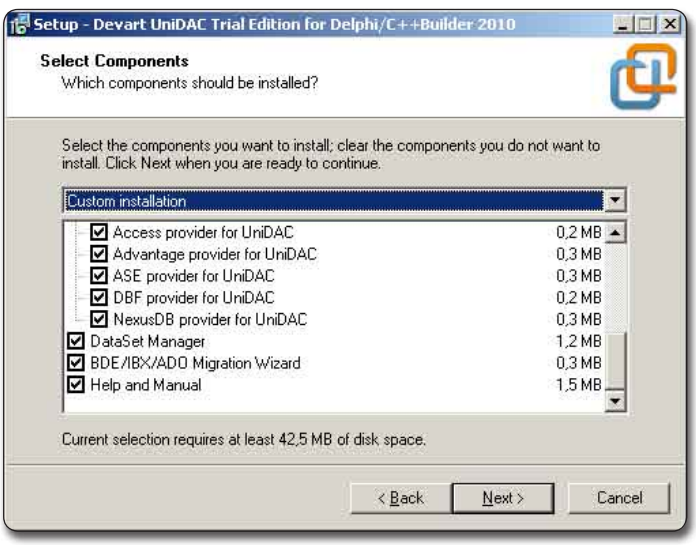



**Figura 8.** Instalação do Unidac

Em params a opção database é o caminho para o banco employee.fdb, que no caso deste exemplo foi feita uma cópia do banco de dados. User_name é sysdba, password é masterkey e o SQLdialect é 3.

O SQLDataset ligado a esse SQLConnection tem na propriedade connection name SQLConnection1 e como instrução SQL tem "select * from CUSTOMER".

A conexão ODBC com o banco de dados Firebird pode ser criada abrindo-se o painel de controle do Windows e no "Administrador de fontes de dados ODBC" criar uma nova, usando o driver ODBC do Firebird, como mostra a **Figura 9,** nela também se mostra necessário colocar login, senha, caminho até o banco de dados e um nome para essa conexão, chamada aqui de FirebirdODBC. Esse nome será visto e usado tanto no BDE como no ADO.

Abrindo o BDE no painel de controle é possível ver o FirebirdODBC, que foi automaticamente criado a partir das conexões ODBC como pode ser visto na **Figura 10**.

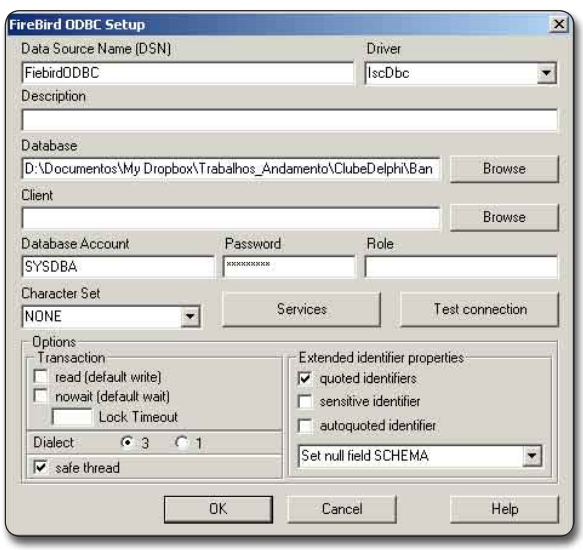



**Figura 9.** Configurando a conexão

**Nota**

O BDE é um mecanismo muito antigo de conexão com banco de dados, presente no Delphi desde a primeira versão. Seu uso não é encorajado a não ser em questões de integração e interoperabilidade com sistemas legados ou migração. Para fins didáticos o BDE foi citado nesse artigo.

**Nota**

Existe também a coleção ZeosLib, open-source, cuja versão 7 pode ser baixada e instalada, mas ainda se encontra em fase alpha. Vale a pena mencionar sua existência, mas não necessariamente será fácil instalar e fazer funcionar no Delphi XE.

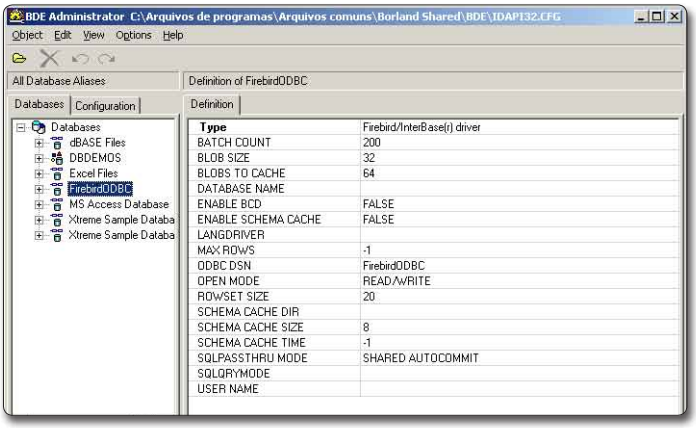



**Figura 10.** Como o FirebirdODBC aparece no BDE

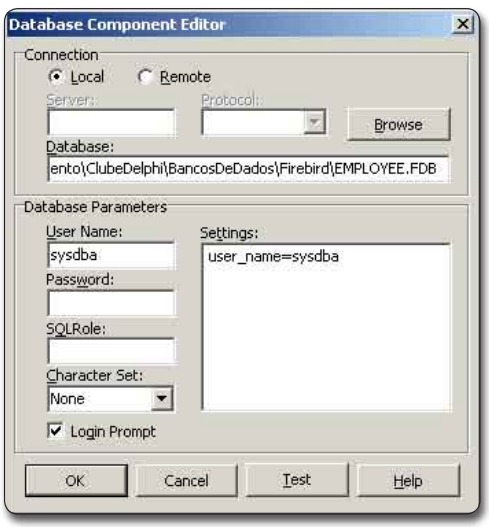



**Figura 11.** Conexão com IBX

Um componente TDatabase da paleta BDE pode ser adicionado ao formulário, tendo seu DatabaseName como TesteFirebirdODBC e o AliasName como FirebirdODBC. Com um duplo clique no TDatabase se acessa essa janela de configuração.

A configuração completa, vista no parameter overrides, é esta:

```
DATABASE NAME=
USER NAME=sysdba
ODBC DSN=FirebirdODBC
OPEN MODE=READ/WRITE
BATCH COUNT=200
LANGDRIVER=
MAX ROWS=-1
SCHEMA CACHE DIR=
SCHEMA CACHE SIZE=8
SCHEMA CACHE TIME=-1
SQLPASSTHRU MODE=SHARED AUTOCOMMIT
SQLQRYMODE=
ENABLE SCHEMA CACHE=FALSE
ENABLE BCD=FALSE
ROWSET SIZE=20
BLOBS TO CACHE=64
BLOB SIZE=32
PASSWORD=masterkey
```

## Nota do DevMan

Um teste de performance mais metódico e imparcial consistiria em ter, para cada teste, uma máquina limpa, recém formatada, igual em configuração de software e hardware para cada teste. Os testes deveriam ser repetidos múltiplas vezes e o primeiro teste deveria ser sempre desprezado visto que no primeiro acesso os dados serão armazenados no cache do SGBD para posterior consulta, e isso tornaria as consultas subsequentes mais rápidas. Não se pode desconsiderar também o cache da própria HD. É desnecessário afirmar que as tabelas, quantidades de registros, índices e a consulta em si devem ser idênticos. Além do teste de consulta um bom teste de performance testaria também a inserção de uma quantidade enorme de registros, calculando o tempo total e o tempo de cada um, e uma atualização de um número enorme de registros. Ambos podem ser feitos em laços do tipo for i := 1 to 1000000 por exemplo.

**Nota**

Se ao tentar acessar o banco de dados Firebird através de um componente IBX ocorrer a mensagem de erro "unavailable database" esse problema pode ser devido à versão do Firebird: pode estar faltando a biblioteca gds32.dll para retro-compatibilidade. Um outro motivo desse erro pode ser o modo de conexão: em vez de colocar local o correto seria colocar remoto, via TCP, e o nome do server colocar o host ou ip do servidor, ou colocar localhost se for a própria máquina de desenvolvimento. Entretanto o acesso ao Firebird através de IBX não é mais recomendado, pois IBX foi feito para acessar Interbase. Para acessar Firebird o correto seria usar DBX ou algum outro componente próprio.

Adicionando um TTable ou TQuery para acessar a tabela customer deve-se colocar TesteFirebirdODBC na propriedade DataBaseName. Este é o nome do database que foi escolhido no componente Tdatabase. Para se conectar ao Firebird com IBX é necessário colocar no formulário um componente IBDatabase, um IbTransaction e um IbDataset ou IbQuery e fazer as devidas ligações entre eles. Com um duplo clique no IBDatabase aparece a tela da **Figura 11**, que permite a conexão com o banco.

Para se criar uma conexão ADO+ODBC deve-se colocar um AdoConnection da paleta dbgo e um AdoDataset. Um duplo clique no AdoConnection abre uma tela perguntando se será selecionado um arquivo udl (que contém a connection string) ou se uma connection string será montada. Ao montar a connection string deve-se escolher o OLEDB provider para drivers ODBC como mostra a **Figura 12**.

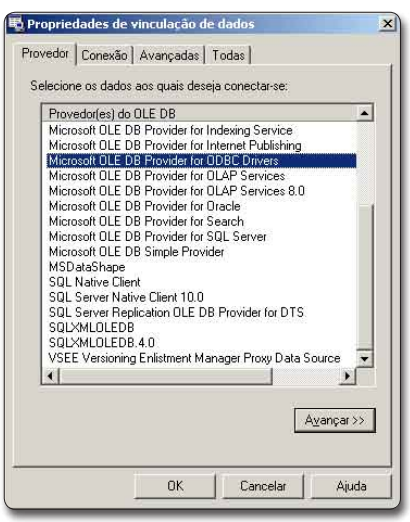



**Figura 12.** ADO + ODBC

O nome da fonte de dados é o nome FirebirdODBC anteriormente escolhido. Para acessar o SGDB Firebird usando OLEDB um driver (ou provider) OLEDB deve ser instalado. Na seção de links há dois providers para se escolher. O usado nesse artigo é o IBProvider, que é trial mas pode ser usado por tempo suficiente para se testar o exemplo aqui proposto.

O procedimento para conexão ADO+OLEDB é o mesmo que o ADO+ODBC, exceto pelo detalhe de que em vez de se selecionar o provider para ODBC se seleciona o IBProvider, conforme visto na **Figura 13**.

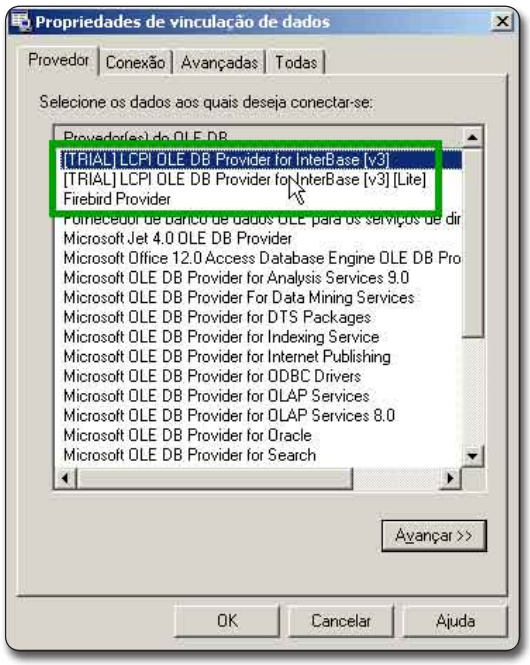



**Figura 13.** Escolhendo o IBProvider

Nos parâmetros de conexão é importante marcar a opção "Enable Automatic Transaction" para que selects possam ser feitos em uma transação que é criada automaticamente, assim os TPersistentFields podem ser adicionados normalmente ao AdoDataset. É importante escolher o tipo correto do Server, no caso Firebird 2.x.

Para conexão via Unidac todos os parâmetros estão no próprio componente TuniConnection e podem ser editados no object inspector como mostra a **Figura 14**.

Os parâmetros são Username, Password, Database (é o caminho até o arquivo fdb desde a raiz), Server (o servidor onde o banco se encontra) e o ProviderName é "Interbase", que é compatível com Firebird.

Os componentes necessários são um TUnidatabase, um UniQuery e um InterbaseUniProvider, nas paletas UniDac e UniDac Providers respectivamente.

O importante é que no teste todos os componentes do tipo Database ou Connection estejam com loginprompt = false e connected = false. A **Figura 15** mostra um formulário com todos os componentes configurados para acessar o Firebird. Da esquerda para a direita tem-se: DBX, BDE, IBX, ADO + ODBC, ADO+OLEDB e Unidac.

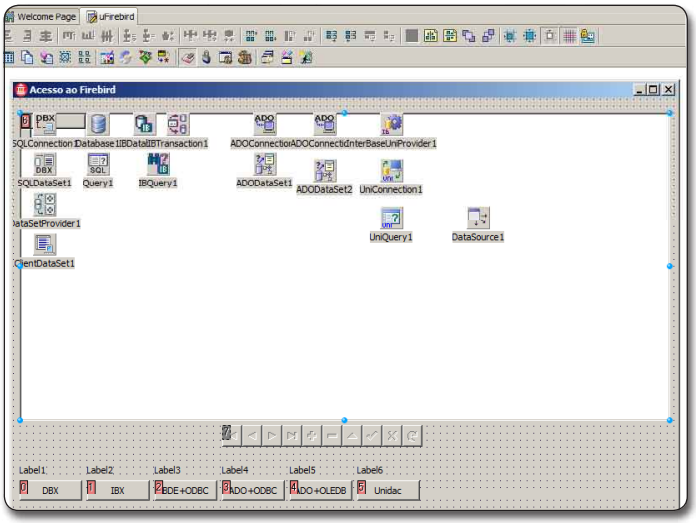



**Figura 15.** Todos os componentes

Os botões de 1 a 6 farão as conexões, respectivamente, usando DBX, IBX, BDE+ODBC, ADO+ODBC, ADO+OLEDB e Unidac. O DbNavigator1 abaixo do DbGrid1 pode ser usado para testar a inclusão, exclusão e atualização dos dados, lembrando que o Clientdataset precisa executar o método ApplyUpdates para que as alterações sejam efetivamente refletidas no banco.

Os componentes Query e IBQuery não permitirão exclusões, inserções ou updates sem os respectivos componentes UpdateSql. Serão somente leitura.

O teste simplesmente tem 6 botões e 6 conjuntos de componentes de acesso. Cada conjunto fará a conexão com o employee.fdb de uma maneira diferente, mostrará os dados no dbgrid e medirá o tempo de abertura. O employee.fdb, mais precisamente a tabela customer que está sendo acessada, é muito trivial. Tem poucos dados. Como projeto particular de pesquisa crie uma rotina para popular essa tabela com um milhão de registros e aí sim veja a velocidade de acesso, e também a de escrita, de cada componente.

Nos próximos bancos de dados analisados é perfeitamente possível fazer um acesso via BDE+ODBC, mas não será necessário,

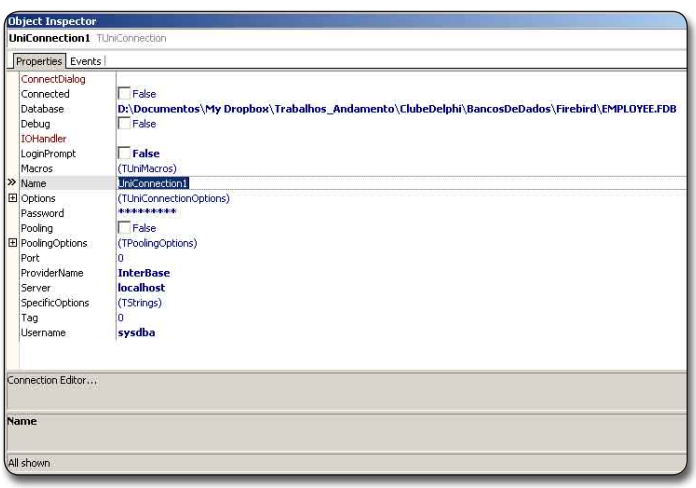



**Figura 14.** Propriedades do UniConnection

tampouco é recomendável fazer isso. O leitor, se desejar, pode fazer um teste com um banco de dados Firebird maior.

A **Listagem 1** mostra o código desse teste. A maneira de se conectar a cada banco pode ser vista no código fonte deste artigo.

Junto com o código fonte desse artigo serão disponibilizados os arquivos dsn que são as configurações de conexão ODBC e os arquivos UDL que são as strings de conexão ADO.

Como prova de conceito já foi mostrado que praticamente qualquer banco de dados pode ser acessado pelo antigo BDE com ODBC ou mesmo pelo ADO, por isso nos próximos testes o foco será conexão via DBX e Unidac. A maneira de se conectar muda pouco de banco para banco.

## Conectando-se ao My SQL

O processo para se acessar um banco de dados MySql pelo Delphi é análogo ao processo usado para se acessar Firebird. Para se conectar com DBX deve-se colocar o quarteto SQLConnection, SQLDataset, Provider e ClientDataset e fazer as devidas ligações.

No SQLConnection a propriedade params deve ser editada. O parâmetro HostName deve ser definido como localhost se o banco estiver na máquina de desenvolvimento ou o nome (ou ip) da máquina onde se encontra o banco.

User_Name é root ou algum outro que tenha sido criado e Password é a senha definida para esse usuário ou a senha de root definida durante a instalação. O DriverName será MySql e o banco de dados será o banco de exemplo employees, como na **Figura 16**.

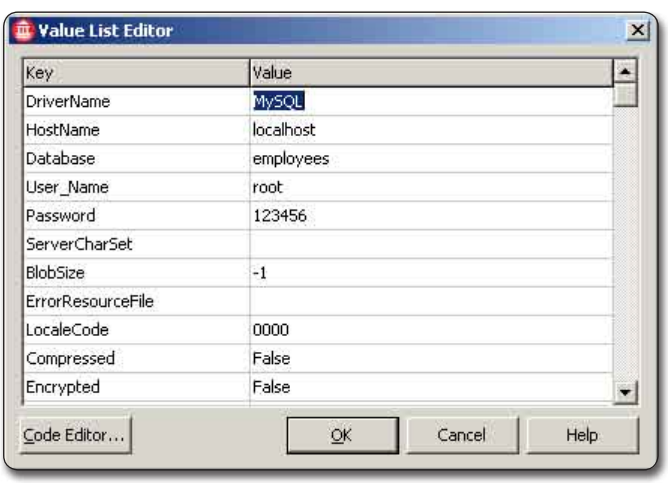



**Figura 16.** Params do SQLConnection

O banco de dados employees baixado do site do MySql, na seção de documentação, é diferente do banco de dados employee.fdb do Firebird. Nesse não há uma tabela customer e a tabela que será testada é a tabela de empregados, employees. Além do banco employees há outros bancos de dados de exemplo junto com a documentação do MySql. Os arquivos disponíveis são as tabelas em arquivos de texto .sql e os "dumps", arquivos grandes com as instruções de insert para popular a tabela. O banco deve ser criado e os dumps executados em alguma ferramenta de gerenciamento do MySql, como o PHPMyAdmin.

**Nota**

A Devart também disponibiliza os componentes ibdac para acesso exclusivo a interbase e Firebird. Além disso há o dbxida que é um driver DBX para interbase e Firebird feito pela Devart.

Para se conectar ao MySql com o Unidac é necessário um Uniconnection, um MySqlUniProvider e um UniQuery. A configuração desses componentes é muito fácil. Assim como no caso do firebird, basta dar um duplo clique no Uniconnection que a janela da **Figura 17** aparecerá.

Deve-se colocar como propriedade provider o valor MySql. Para conexão com o ADO basta instalar o ODBC Connector do MySql, disponível na seção de links, e criar a conexão ODBC com o banco MySql através das "Fontes de dados ODBC" no painel de controle>ferramentas administrativas ou criar um arquivo dsn. Tanto a conexão como o arquivo podem ser criados através do próprio AdoConnection clicando-se em "usar sequência de conexão" e "criar" (**Figura 18**).

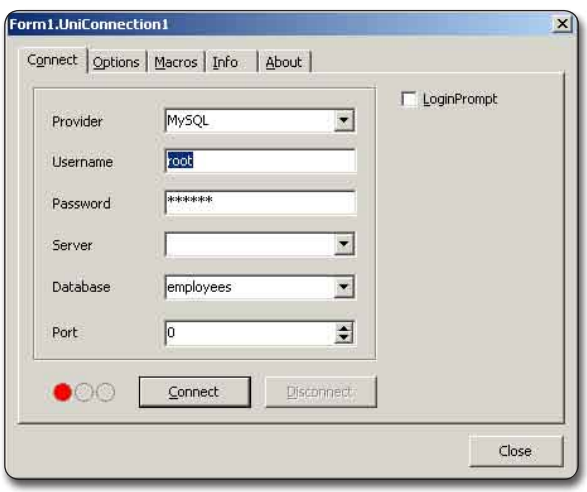



**Figura 17.** Configuração do MySql

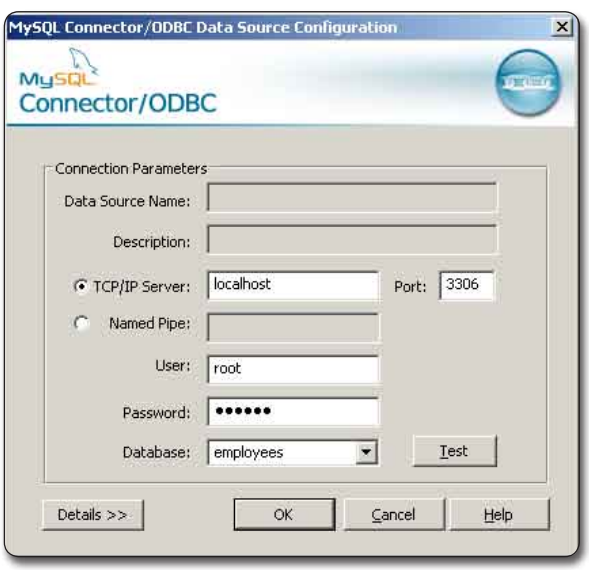



**Figura 18.** Janela de configuração do driver ODBC

A connection string gerada automaticamente e armazenada na janela foi esta:

```
Driver=MySQL ODBC 5.1 Driver;SERVER=localhost;UID=root;PWD=123456;DATABASE=
employees;PORT=3306;COLUMN_SIZE_S32=1
```

Esta string pode ser montada pelo programador e digitada no campo apropriado "sequência de conexão". Como pode ser visto na **Figura 19** neste exemplo será feito um pequeno benchmark também, medindo o tempo de abertura de cada tipo de componente. É bom lembrar, no entanto, que a performance depende também das políticas de cache de cada componente, de quantos dados estão sendo trazidos, se há índices ou não e se há consultas aos metadados ou não entre vários outros fatores.

Da esquerda para a direita vê-se os componentes SQLConnection e SQLDataset da tecnologia DBExpress, seguidos dos UniDatabase, provider e query da Devart e terminando com AdoConnection AdoDataset.

O código é muito semelhante ao anterior e pode ser visto na **Listagem 2**. Faça os testes começando pelo Unidac, depois Ado e deixe o DbExpress por último.

A conclusão dos testes é que o DBExpress dessa vez foi muito mais lento do que o Unidac e mais lento até do que o ODBC. Até BDE seria melhor nesse caso. É possível melhorar a performance por colocar a propriedade GetMetaData do SQLDataset para false.

O metadata é o conjunto de dados sobre dados, ou a estrutura do banco de dados. Em uma consulta simples não é necessário trazer os metadados, nem faz sentido. Faria sentido em queries dinâmicas onde se necessita consultar os índices para se montar queries otimizadas, exatamente o ambiente do trio SQLDataset, provider e ClientDataset.

Segundo o help do Delphi, o GetMetaData false, em consultas, pode aumentar a velocidade das consultas porque nenhuma infor-

**Nota**

Se no computador de desenvolvimento existir mais versões anteriores do Delphi instaladas é possível que o DBX não consiga se conectar com o MySql. Isso acontece devido a conflitos de versão entre o driver DBX e a biblioteca client LibMySQL.dll. Se isso acontecer é necessário que a LibMySql seja atualizada na pasta System32 do Windows. Se na variável de ambiente path estiver o caminho para alguma versão anterior do Delphi e nesse caminho existir uma LibmySql.dll desatualizada o problema também ocorrerá, por isso é necessário assegurar-se que o caminho da versão mais atual do Delphi esteja a frente de todos no path. Na seção de links tem um post mostrando como corrigir esse problema. A versão do MySql usada nessa edição é a 5.0.x mas o DBX do Delphi 2010 e XE também funciona com a versão 5.1.x. A versão 5.5 deve ser evitada, pois ainda não é compatível, não existe driver DBX "oficial" compatível. A Devart possui, além dos componentes de acesso nativo, um driver DBX com suporte a MySql 5.5, é uma alternativa interessante.

**Listagem 1.** Conectando-se de várias formas diferentes ao employee.fdb

```
procedure TForm1.Button1Click(Sender: TObject);
var
 T: integer;
begin
 if ClientDataSet1.Active then
    ClientDataSet1.Close;
 t := GetTickCount;
 ClientDataSet1.Open;
 t := GetTickCount -t;
 Label1.Caption :=  IntToStr(t);
 DataSource1.DataSet := ClientDataSet1;
end;
procedure TForm1.Button2Click(Sender: TObject);
var
 T: integer;
begin
 if Query1.Active then
    Query1.Close;
 t := GetTickCount;
 Query1.Open;
 t := GetTickCount -t;
 Label2.Caption :=  IntToStr(t);
 DataSource1.DataSet := Query1;
end;
procedure TForm1.Button3Click(Sender: TObject);
var
 T: integer;
begin
 if IBQuery1.Active then
    IBQuery1.Close;
 t := GetTickCount;
 IBQuery1.Open;
 t := GetTickCount -t;
 Label3.Caption :=  IntToStr(t);
 DataSource1.DataSet := IBQuery1;
end;

procedure TForm1.Button4Click(Sender: TObject);
var
 T: integer;
begin
 if ADODataSet1.Active then
    ADODataSet1.Close;
 t := GetTickCount;
 ADODataSet1.Open;
 t := GetTickCount -t;
 Label4.Caption :=  IntToStr(t);
 DataSource1.DataSet := ADODataSet1;
end;
procedure TForm1.Button5Click(Sender: TObject);
var
 T: integer;
begin
 if ADODataSet2.Active then
    ADODataSet2.Close;
 t := GetTickCount;
 ADODataSet2.Open;
 t := GetTickCount -t;
 Label5.Caption :=  IntToStr(t);
 DataSource1.DataSet := ADODataSet2;
end;
procedure TForm1.Button6Click(Sender: TObject);
var
 T: integer;
begin
 if UniQuery1.Active then
    UniQuery1.Close;
 t := GetTickCount;
 UniQuery1.Open;
 t := GetTickCount -t;
 Label6.Caption :=  IntToStr(t);
 DataSource1.DataSet := UniQuery1;
end;
```

mação além da consultada é trazida, mas pode tornar mais lentos os updates, principalmente em ClientDatasets ligados através de providers em SQLDatasets, porque não será possível consultar os índices para montar uma instrução SQL de update otimizada. Também tornará mais lentos os casos de atualizações mestre-detalhe. O ideal seria colocar sempre GetMetaData como false para datasets onde se farão apenas consultas, ou seja, somente leitura, e true caso contrário.

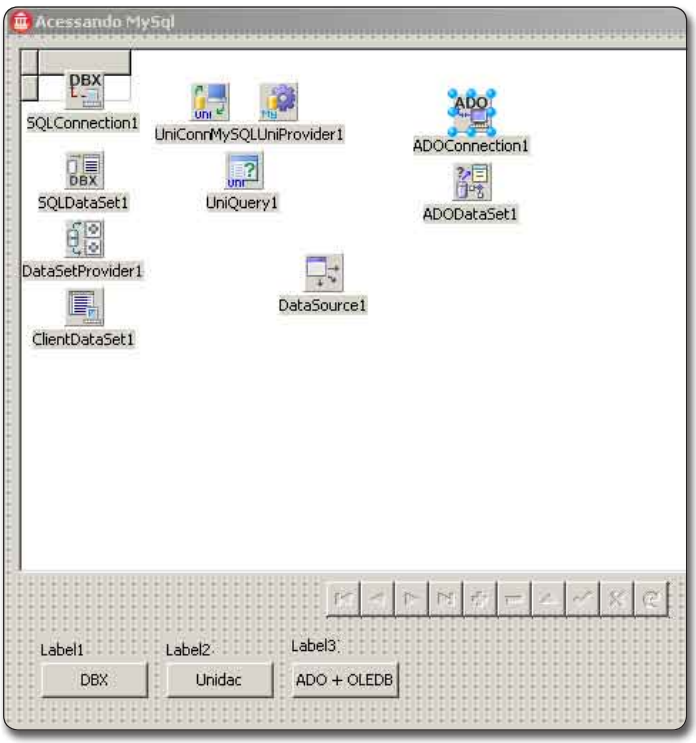



**Figura 19.** Componentes para conexão com MySql

## Conectando-se ao PostgreSQL

Analogamente aos testes com Firebird e MySql, o banco de dados postgre também será acessado com DBExpress, ADO e Unidac. O banco de dados de exemplo escolhido é o world, que pode ser baixado na página de bancos de dados de exemplos.

Esse banco de dados world vem no formato de dump onde, em um arquivo texto, há instruções SQL que executadas criam a tabela e instruções para que as tabelas recém criadas sejam carregadas da entrada padrão. O banco de dados deve ser criado com a ferramenta de modo texto (console) psql Shell.

A **Figura 20** mostra os comandos para criar o database e importar o dump no console. Juntamente com o fonte desse artigo existe um arquivo de backup desse banco de dados, que pode ser restaurado com ajuda da ferramenta PGAdmin.

O PGAdmin é uma interfaces gráfica para administração do banco de dados postgre, assim como o IBConsole é para o Interbase, e o IBOConsole ou IBExpert é para o Firebird. Quem trabalha com MS SQL Server também conhece o Enterprise Manager. O PGAdmin é muito semelhante a essas ferramentas.

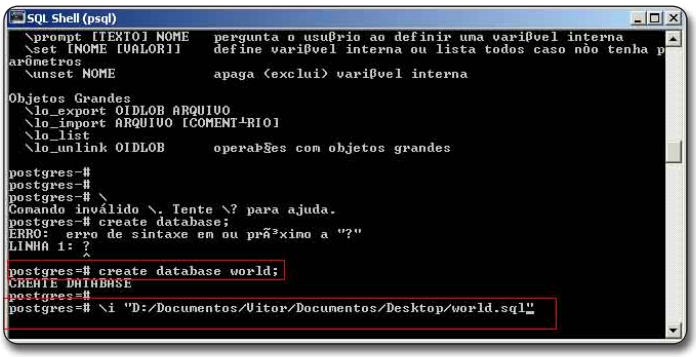



**Figura 20.** Comandos para criar e popular o banco de dados world

**Listagem 2**. Conexão com o banco Employees no MySql

```
procedure TForm1.Button1Click(Sender: TObject);
var
 T: integer;
begin

 if ClientDataSet1.Active then
  ClientDataSet1.Close;

 t := GetTickCount;
 ClientDataSet1.Open;
 t := GetTickCount -t;
 Label1.Caption :=  IntToStr(t);
 DataSource1.DataSet := ClientDataSet1;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
 T: integer;
begin

 if UniQuery1.Active then
  UniQuery1.Close;

 t := GetTickCount;
 UniQuery1.Open;
 t := GetTickCount -t;
 Label2.Caption :=  IntToStr(t);
 DataSource1.DataSet := UniQuery1;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
 T: integer;
begin

 if ADODataSet1.Active then
  ADODataSet1.Close;

 t := GetTickCount;
 ADODataSet1.Open;
 t := GetTickCount -t;
 Label3.Caption :=  IntToStr(t);
 DataSource1.DataSet := ADODataSet1;
end;
```

O comando create database world; criará o banco vazio. Logo depois deve se executar o comando \connect world para se conectar ao banco recém-criado. O comando \i serve para executar uma lista de comandos armazenados em um arquivo, no caso o world.

sql. No exemplo as aspas foram colocadas apenas para destacar o nome do arquivo. O comando deve ser executado sem aspas, e não se deve usar barras invertidas, apenas barras de "divisão" como nos nomes de arquivo em Unix.

Todos os comandos iniciam com uma "contrabarra" \. Caso o arquivo world.sql seja executado antes de se conectar ao banco world as tabelas serão criadas no banco padrão, postgres.

Para se conectar a esse banco, com Unidac, não há segredo. Basta colocar o Uniconnection e um PostgreSQLUniProvider da paleta UniDAC Providers. Setar as propriedades corretamente, com um duplo clique no componente: ProviderName sendo PostgreSQL, Server LocalHost, username postgres, password 123@abc, ou qualquer que seja o password colocado para o usuário postgres durante a instalação.

O world é um banco de dados de países, linguagens e cidades. A maior tabela é a de cidades, já que um país tem várias cidades. A query executada em todos os exemplos será select * from city.

Para se conectar via DBExpress o Delphi não vem com um driver DBX para o PostgreSQL, mas isso não é problema, drivers de terceiros podem ser baixados. O site da Vita Voom tem um driver que pode ser baixado. É apenas demo, mas servirá por tempo suficiente para que se realizem testes. A Devart também possui um driver DBExpress para PostgreSQL.

Com o ADO há duas formas de se conectar, com OLEDB e com ODBC. A maneira de configurar é a mesma, ou seja, as propriedades a serem preenchidas, Server, username, password, databasename são as mesmas dos outros componentes.

Por último, nos testes, o SQLConnection (dbExpress) foi conectado usando o driver da Devart. Clicando em params veem-se as mesmas configurações de sempre: user_name, password, hostname e database. Essa janela de configuração pode ser conferida na **Figura 21**.

O fonte não muda nada em relação aos anteriores. É deixado a cargo do leitor fazer os testes de inserção, update, mestre-detalhe etc. A **Listagem 3** apresenta o código fonte desse exemplo.

Para o driver DBX da Devart funcionar deve-se adicionar no library path do Delphi o caminho de instalação das dcu's da Devart: "C:\Arquivos de programas\Devart\Dbx\PostgreSql\Delphi15", lembrando que o Delphi 14 é o 2010 e o Delphi 15 é o XE.

Da esquerda para a direita na **Figura 22** temos os componentes para acessar o world via DBX, Unidac, OLEDB, ODBC e BDE. Ainda hoje é possível acessar qualquer banco de dados com a BDE, via ODBC, e por mais que seu uso seja desencorajado, ainda hoje muitas empresas a utilizam.

Vale ressaltar aqui que qualquer resultado nesses testes dirão qual é a performance de uma consulta simples, um select. Não dizem nada a respeito de qual componente gerencia melhor o cache ou qual é mais rápido em atualizações.

Para ver qual gera o SQL mais otimizado monitores de SQL ou logs, que são diferentes para cada banco ou componente de acesso, deveriam ser vistos, e o uso de memória acompanhado pelo gerenciador de tarefas do Windows, ou um gerenciador de tarefas mais avançado, como o da Windows Internals.

Além disso, em consultas, qualquer descendente de TDataset pode ser ligado a um provider e um ClientDataset: os Uniquerys, os AdoDatasets e até mesmo um TQuery da BDE. Isso não garante porém, que as instruções SQL de update, insert e delete serão geradas corretamente ou da maneira mais otimizada possível. Mas para a maioria dos casos isso é verdadeiro.

Cabe ao desenvolvedor testar essas combinações, entretanto esse conjunto de exemplos desse artigo já fornece um bom esqueleto para que se façam novos testes.

## Dicas de performance

Não importa qual o banco de dados usado em um projeto ou qual seja o componente de acesso, há algumas dicas de performance que podemos considerar:

1. Nunca faça "select * from tabela", por vários motivos:
a Com * todos os campos serão trazidos, até mesmo os desnecessários;
b. Com * um novo campo recém-adicionado na tabela será trazido, mesmo que não seja necessário;
c. Você pode perder o controle dos campos que traz simplesmente por não referenciá-los por nome;
d. Em alguns bancos de dados os índices não são usados com select *;
2. Crie índices para campos relevantes. Campos relevantes são aqueles que nunca serão deixados null e que são diferentes em quase todos os registros, por exemplo "nome";
3. Não crie índices para campos que possuirão poucos valores diferentes, como Estado ou Sexo;
4. Um conjunto grande de campos adicionados em uma tabela poderia ser colocado em uma tabela separada, com uma relação de 1 – 1 caso o preenchimento desses campos não seja obrigatório e seja null na maioria dos casos. Evite colocar muitos campos de preenchimento opcional;
5. Faça periodicamente as manutenções recomendadas pelo fabricante do seu SGBD, como um índex defrag por exemplo;
6. Use stored procedures, elas são capazes de fazer uma grande quantidade de tarefas, até processamentos pesados, no servidor do banco de dados e também porque o comando SQL para disparar uma procedure é relativamente mais curto do que um select cheio de filtros ou um update complexo;
7. Traga somente os registros relevantes e desses, apenas os campos relevantes. É melhor fazer perguntas ao usuário na interface antes de trazer os campos, para trazê-los filtrados, do que trazê-los aos montes para que o usuário os escolha. Obviamente isso depende do contexto. Num caso de itens de um pedido ou de uma nota fiscal, casos típicos de composição (ou mestre-detalhe) traga todos os itens;
8. Para índices compostos procure colocar os nomes dos campos na mesma ordem em que o faria em um where. Por exemplo, se em uma consulta de produto o filtro fosse "where família=:família and tipo=:tipo" procure criar o índice (família,tipo) se essa consulta for feita muitas vezes e o número de produtos for muito grande;
9. Nunca esqueça do where;
10. Servidor de aplicação (em ambientes web ou data snap) no mes-

mo servidor do banco de dados podem incrementar a velocidade de transmissão dos dados de uma consulta, mas podem prejudicar o processamento, pois aplicação e SGBD competirão por recursos como disco, memória e processador. As duas arquiteturas devem ser muito bem testadas pois cada caso é diferente;

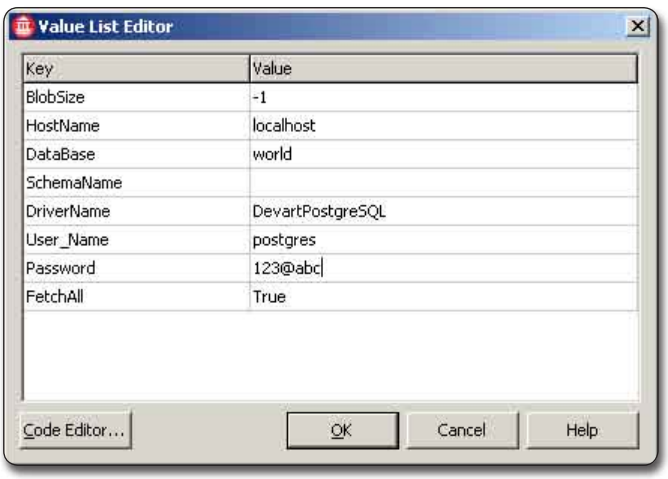



**Figura 21.** Configurando o dbExpress com driver DevartPostgreSQL

11. Em ambientes web, como e-commerces, use cache para todas as páginas que não mudam frequentemente as quais os usuários visualizam antes de estar logados, como as páginas de vitrines, por exemplo. Isso diminuirá as consultas repetidas em um banco;
12. Se o SGBD permitir, particione o banco visando a performance: alguns bancos permitem deixar certas tabelas ou até mesmo os índices em arquivos separados;
13. Banco de dados em HD diferente da HD do sistema operacional não compete com o SO por I/O;
14. Existem dois tipos de tabelas: as de entidades/cadastros, como por exemplo Clientes e Produtos, e existem as tabelas de transações/eventos, como as tabelas de pedidos, carrinhos abandonados, sessões, logs. Se possível "esvazie" periodicamente as tabelas de transações movendo todas as transações anteriores a uma data de corte para outras tabelas, históricas, preferencialmente até em outro banco de dados, que sirva somente para relatórios estatísticos. Assim as tabelas

**Nota**

O BDE não suporta Unicode ou UTF8, e esse é só um dos vários motivos pelos quais ele é desencorajado hoje: o suporte multilíngue. Sendo assim o encoding, collation e character type do banco de dados world no postgre deve ser win1252 para que seja suportado pelo BDE.

**Listagem 3.** Conexões possíveis com o postgreSQL

```
procedure TForm1.Button1Click(Sender: TObject);
var
 T: integer;
begin

 if ClientDataSet1.Active then
  ClientDataSet1.Close;

 t := GetTickCount;
 ClientDataSet1.Open;
 t := GetTickCount -t;
 Label1.Caption :=  IntToStr(t);
 DataSource1.DataSet := ClientDataSet1;

end;

procedure TForm1.Button2Click(Sender: TObject);
var
 T: integer;
begin

 if UniQuery1.Active then
  UniQuery1.Close;

 t := GetTickCount;
 UniQuery1.Open;
 t := GetTickCount -t;
 Label2.Caption :=  IntToStr(t);
 DataSource1.DataSet := UniQuery1;

end;

procedure TForm1.Button3Click(Sender: TObject);
var
 T: integer;
begin

 if ADODataSet1.Active then
  ADODataSet1.Close;

 t := GetTickCount;
 ADODataSet1.Open;
 t := GetTickCount -t;
 Label3.Caption :=  IntToStr(t);
 DataSource1.DataSet := ADODataSet1;

end;

procedure TForm1.Button4Click(Sender: TObject);
var
 T: integer;
begin

 if ADODataSet2.Active then
  ADODataSet2.Close;

 t := GetTickCount;
 ADODataSet2.Open;
 t := GetTickCount -t;
 Label4.Caption :=  IntToStr(t);
 DataSource1.DataSet := ADODataSet2;
end;

procedure TForm1.Button5Click(Sender: TObject);
var
 T: integer;
begin

 if Query1.Active then
  Query1.Close;

 t := GetTickCount;
 Query1.Open;
 t := GetTickCount -t;
 Label5.Caption :=  IntToStr(t);
 DataSource1.DataSet := Query1;

end;
```

"quentes" ficarão sempre com um mínimo de registros, apenas os mais recentes. Relatórios completos podem ser tirados fazendo um Union entre as tabelas históricas e as "quentes".

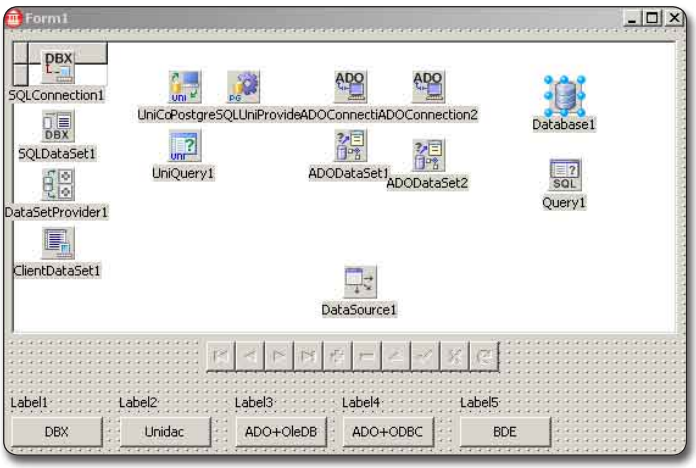



**Figura 22.** Todos os componentes de acesso

## Conclusão

O objetivo desse artigo foi fornecer um guia inicial de bancos de dados relacionais gratuitos para que o desenvolvedor possa conhecer, escolher e saber conectar-se a cada um deles. Não foi possível, em apenas um artigo, por questões de espaço e de escopo, abordar a linguagem SQL.

Espera-se que ao final da leitura desse artigo o leitor possa acessar qualquer dos bancos de dados mencionados através de qualquer um dos mecanismos conhecidos, e que possa acessar ainda outros que surgirão da mesma forma: pesquisando por um driver adequado.

Com o código fonte deste artigo podem ser encontrados exemplos de como se conectar a cada tipo de banco de dados usando tecnologias diferentes. Não foi possível em um artigo só falar sobre todos os bancos de dados gratuitos possíveis de se conectar com o Delphi. Ainda faltou o Oracle Express, o SQL Server Express e uma série de outros. Existem também os forks de bancos de dados conhecidos, como o MariaDB e o Yaffil, que é outro fork, russo, do Interbase, paralelo ao Firebird.

O que dá para perceber é que o Delphi tem os componentes para acessar a maioria dos bancos de dados mais conhecidos ou populares. Mesmo que não tenha nativamente um driver de terceiros pode ser adquirido. Além disso, qualquer banco de dados que tenha um driver OLEDB ou ODBC pode ser acessado pelo ADO ou até pela BDE. Sendo assim, virtualmente qualquer banco de dados pode ser acessado por um projeto em Delphi.

Mesmo que não se encontre nenhum componente de acesso a um determinado banco de dados, todos eles têm alguma biblioteca client, e essa possui uma SDK que mostra como acessar sua API e usar o banco de dados. O código seria bastante extenso, mas conceitualmente seria apenas um punhado de classes e métodos para acessar algumas dlls e controlar alguns recursos, exatamente o que os componentes de acesso a dados fazem nos bastidores.

**Vitor Luiz Rubio**
*vitorrubio@gmail.com*
*http://blog.vitorrubio.com.br*
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP.
Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais.
Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

**Blogs do autor**
*http://www.vitorrubio.blogspot.com/*
*http://twitter.com/vitorrubio*

**Firebird**
*http://www.Firebirdsql.org/*

**Firebird ODBC**
*http://www.Firebirdsql.org/index.php?op=files&id=odbc*

**Firebird OLEDB IBProvider**
*http://www.ibprovider.com/eng/*

**IBOLEDB**
*http://iboledb.sourceforge.net/*

**IBObjects**
*http://www.ibobjects.com/ibodownload.html*

**MySql**
*http://www.mysql.com/downloads/mysql/*

**MySql Sample Database**
*http://dev.mysql.com/doc/index-other.html*

**MySql Connector ODBC**
*http://dev.mysql.com/downloads/connector/odbc/*

**dbExpress suporte ao MySQL no Delphi 2010 e XE**
*http://www.andreanolanusse.com/blogpt/dbexpress-suporte-ao-mysql-no-delphi-2010-e-xe/*

**PostgreSql**
*http://www.postgresql.org.br/*

**Postgre Sample Databases**
*http://pgfoundry.org/projects/dbsamples/*

**Driver DBExpress Vita Voom**
*http://www.vitavoom.com/downloads/*

**Driver DBExpress Devart**
*http://www.devart.com/dbx/postgresql/download.html*

**Unidac**
*http://www.devart.com/unidac/download.html*

**Zeos**
*http://zeos.firmos.at/portal.php*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Lookup Personalizado

## Construindo um componente para substituir os tradicionais lookups

**Vitor Luiz Rubio**

Em RAD (Rapid Application Development), o tipo de projeto mais comumente desenvolvido em Delphi, VB ou C#, é muito comum o uso de lookups. Uma característica da programação RAD é que, da maneira como tradicionalmente os programadores fazem, ela é bem "procedural" e orientada a eventos. Além disso o foco, ou a direção do desenvolvimento é altamente dependente e inspirada em tabelas. Nesse ambiente, sempre que é necessário cadastrar um registro em uma tabela, por exemplo produtos, e necessita-se de uma chave primária de outra tabela, por exemplo família ou gênero, o programador tem três opções:

- Colocar um lookup para abrir a tabela a ser relacionada, mostrar os registros e trazer a chave primária do registro selecionado no registro que está sendo editado;
- Abrir um formulário de consulta, geralmente "inteligente" que fará perguntas para o usuário preencher campos e com isso montar em tempo de execução uma consulta que traga o mínimo de dados possível do banco, sendo assim bem mais performático;
- Subentender que o usuário já conhece de cor o código da família ou gênero e deixar que ele preencha manualmente, ou consulte primeiro depois preencha, fatalmente incorrendo em erro humano e em violação de integridade referencial caso essa regra não tenha sido imposta no próprio banco.

Por incrível que pareça existem grandes empresas, com ERPs grandes, conceituados e caros que usam a abordagem 3.

Não é intenção do artigo condenar o uso dos lookups tradicionais. Eles têm o seu lugar:

- Protótipos de programas, apresentações para clientes;
- Bases de dados com pouquíssimos valores, e valores fixos, como estados e ICMS;
- Programas em início de desenvolvimento;
- Programadores em início de carreira.

Os três grandes defeitos do lookup, não se restringindo apenas a esses três, são os seguintes:

- Só funcionam com seu dataset aberto, perdendo seus dados quando este está fechado;
- Com um número muito grande de registros eles ficam com uma superfície de seleção (a janela ou aba que se abre clicando-se na seta) muito grande, dificultando a vida do usuário;
- Geralmente não é possível impor filtros nos dados que preenchem o lookup, apenas definir quais colunas serão trazidas. As "linhas", ou registros da tabela, vêm todas;

Esses três problemas fazem com que o lookup se torne inconveniente e lento. A lentidão passa a ser cada vez mais perceptível com o crescimento do tamanho das tabelas no banco de dados.

## Resumo DevMan

***De que trata o artigo:***

Este artigo trata dos tradicionais lookups ou lookupfields, seus problemas de performance, vantagens e desvantagens e sugere um modo diferente de se substituir lookups ainda mantendo suas facilidades.

***Para que serve:***

Lookups servem para se procurar em uma tabela, através de algum atributo conhecido, a chave primária de um registro para que seja incluída em outra tabela. Serve para escolher o "mestre" do "detalhe" ou as duas "partes" de uma entidade relacional. Por exemplo, pode ser usado em uma operação de venda tanto para selecionar o cliente ou a forma de pagamento, também como para selecionar os produtos.

***Em que situação o tema é útil:***

Um lookup é útil sempre que surge a necessidade de se gravar um campo do tipo chave estrangeira em um registro de uma tabela, sendo essa chave estrangeira uma chave primária em outra tabela, sem ter a necessidade de saber de antemão qual é esta chave, mas sim visualizando outros atributos importantes da tabela relacionada. Quando essa tarefa se torna lenta, sua utilização precisa ser revista.

***Lookup Personalizado:***

A verdade é que não é uma boa prática usar os tradicionais lookups e seus derivados. Normalmente eles são usados desde meados do Delphi 1 e BDE, e devem ser associados com um Dataset (Table, Query, qualquer um) que traga os dados da tabela a ser relacionada. O maior problema é que geralmente se traz todos os dados. A lentidão disso é proporcional à quantidade de lookups em um formulário, o número de tabelas que eles abrem e o tamanho dessas tabelas. Esse artigo mostra como criar um componente reutilizável que cria um lookup "inteligente" que trará os resultados do banco de dados apenas depois de algumas letras inseridas e que manterá um "cache" para consultas futuras. Ele mostrará vários campos e poderá fazer consultas do tipo "like" também.

Se tornando pior ainda quando o servidor de banco de dados está em um servidor numa rede local e o cliente em outro computador desta mesma rede. Pior ainda se o banco de dados estiver hospedado em um serviço de hosting qualquer, o uso de lookups neste caso é impraticável.

Com certeza há saídas para este problema. Uma delas é usar clientdatasets como uma espécie de cache, que serão populados na inicialização do sistema e não mais se conectarão ao banco de dados. É a prática mais comum. Outra abordagem é colocar ComboBoxes comuns, não-lookups ou não-data-awares, e popular essas caixas de listagem com os valores na inicialização do sistema. Ambas as práticas podem tornar o sistema lento na inicialização.

Outro problema de carregar os dados na inicialização é que caso novos registros sejam adicionados a essas tabelas por outros usuários, em outras estações, os dados não serão atualizados nesses datasets que já estão abertos e populados servindo de cache. Seria necessário implementar um botão de "refresh" ou fechar e abrir novamente o sistema. Nenhuma das duas opções parece cômoda para o usuário.

O meio termo é criar um formulário genérico de consulta que possa ser configurado para consultar qualquer tabela, trazendo os resultados da consulta em uma lista ou grid e devolvendo ao seu "chamador" o código da chave primária do registro selecionado.

**Nota**

Como os lookups geralmente necessitam de duas colunas, a chave primária e outra que seja humanamente possível de se identificar um registro, não há necessidade de se popular os datasets dos lookups com "select * from tabela", e este é uma das piores práticas de todas no desenvolvimento de sistemas, e é um dos principais motivos de lentidão, visto que todos os campos são trazidos.

## Nota do DevMan

Quando um sistema é desenvolvido puramente orientado a objetos e seguindo todos os padrões de projeto e boas práticas dificilmente usa-se lookups. São usadas classes das camadas de persistência para trazer algum objeto procurado, e este é apresentado usando-se uma GUI (form) que seja ligada a ele através de um MVC, por exemplo. Não se fala de chave primária e chave estrangeira nesse contexto, mas o objeto encontrado e trazido (por exemplo a família para o produto) é relacionado com o objeto que está sendo criado ou modificado (o produto).

## Nota do DevMan

RAD é um modelo de desenvolvimento de software maravilhoso, mas tem suas limitações. Sem o devido cuidado o código pode virar uma bagunça, com regras de negócio e acesso a dados espalhados em eventos de controle da interface com o usuário. RAD é recomendado em projetos pequenos, rápidos ou em protótipos. Não é objetivo deste artigo criticar o modelo RAD e muito menos a POO, mas é necessário deixar claro que as técnicas apresentadas neste artigo se aplicam muito mais a RAD, e violam vários dos preceitos ou boas práticas da POO. POO é usada e muito bem vinda tanto na construção de um lookup como de qualquer outro componente. São usados conceitos de herança, polimorfismo, associação, o acesso a dados é delegado para uma classe responsável por isso (o dataset), mas mesmo assim lookups são componentes comumente usados em RAD. Quem usa POO a fundo com padrões de projeto e boas práticas escolhe outras maneiras de relacionar seus objetos e representar essa relação na GUI.

Esta é a técnica que muitos programadores usam. Mas cada vez que o formulário é chamado a consulta é refeita.

O ideal seria manter um cache de acordo com o que foi buscado, e refazer a consulta no banco de dados apenas se a busca for diferente. Essa regra impõe outra: o usuário deve digitar um mínimo de caracteres (um pedaço do nome, CPF ou família de produtos) para que a consulta seja feita e possa funcionar como um "autocompletar".

Para tirar pleno proveito da vantagem do RAD não seria conveniente ter que instanciar e destruir esse formulário de consulta (fazendo ainda as devidas conexões) toda vez que for necessário obter um valor de uma tabela mestre para popular uma chave estrangeira. Por isso o inteiro formulário de consulta pode ser encapsulado dentro de um componente. Se esse componente for um Edit com um botão já dispensa a tarefa de adicionar um Edit e um botão de consulta (ou um lookup) para cada campo em um formulário que seja uma chave estrangeira.

Então esse componente terá os seguintes ingredientes:

- Um TButton ou TBitbtn
- Um TEdit
- Um TForm
- Um DbGrid

Conhecimento prévio sobre criação de componentes é altamente necessário. Embora esse componente tenha como propósito ser usado em projetos RAD (para estender sua vida útil talvez), é importante entender a orientação a objetos por trás da criação desse componente.

O componente se chamará TCDLookup. Ele "é um" TEdit, "contém um" TButton, TForm, Dbgrid etc e "usa um" Datasource, DataSet e assim por diante. As expressões "é um", "contém um" e "usa um" foram colocadas entre aspas propositalmente para deixar claro, na orientação a objetos, de quem o componente herda suas funcionalidades principais (quem é sua superclasse), que outros "ingredientes" que o componente precisa conter dentro de si, mas que ele pode criá-los automaticamente na sua criação ou na chamada a seus métodos, e quais são os objetos com os quais o componente se relacionará (ou dependerá) mas que não necessariamente estarão dentro dele.

Lookups se parecem com TEdits com um botão dentro. Para não perder tempo criando um descendente de TEdit que posicione um botão perfeitamente à sua direita, por dentro, e fazer toda a codificação dos eventos desse botão (que não é o escopo desse artigo), será aproveitado o TButtonedEdit da paleta Additional, que é um TEdit que já vem com botão que pode ser usado para propósitos gerais, seja exibição de formulários ou cálculos, consultas e o que mais for necessário. O botão precisa estar com a propriedade visible = true e precisa que uma imagem seja adicionada a ele para que ele seja visível e "perceptível".

## Componentes no Delphi

No Delphi todo componente nada mais é que uma classe, descendente de TComponent. Isso quer dizer que componentes

não precisam ser "colados" em um formulário, mas podem ser instanciados e usados normalmente.

Um aspecto interessante dos componentes é que o construtor deles aceita um parâmetro chamado aOwner. O Owner, ou dono do componente é passado como sendo o objeto que mandou criar o componente. Quando se arrasta um componente (visual ou não) em um formulário o seu "aOwner" passa automaticamente a ser o formulário. O Owner dos formulários é o objeto Application. Cada vez que um owner é associado com um objeto este é adicionado na lista de filhos desse owner (sua propriedade Components). Isso não tem nenhuma relação com os conceitos de parentwindow e childwindow da API do Windows, mas podem ser concomitante, já que quando se coloca um panel em um formulário este passa a ser owner do panel e parent dele ao mesmo tempo.

A semelhança morre aí, pois quando se coloca um Button nesse panel, o owner do Button não é o panel, é o formulário, mas o parent do Button sim é o panel.

Esse mecanismo de owner foi criado para que caso o programador esquecesse, os componentes pudessem ser liberados da memória com a destruição de seu formulário, e que a aplicação como um todo pudesse ser destruída levando com ela todas os formulários.

Quando um componente é instanciado manualmente nada impede que se passe o valor nil em seu owner, contanto que o programador se lembre de destruir esse componente no final do uso.

Outro conceito importante é que para se criar um formulário sem ter um DFM, o construtor automático, create, não pode ser usado. Existe um outro construtor chamado CreateNew que atende essa demanda.

Os controles criados dinamicamente dentro de um formulário que também foi criado dinamicamente devem definir o formulário como seu parent e seu tamanho, posição e alinhamento devem ser definidos manualmente.

Para o aprendizado da criação de um componente desse tipo se tornar mais didático o primeiro passo seria criar uma classe para uma consulta genérica. Essa classe deve construir um comando SQL e se conectar a um ClientDataset para executar esse comando. O ClientDataset deve ser configurado de maneira a poder executar esse comando. O ClientDataset é uma boa escolha porque pode ser ligado a um servidor DataSnap e ser assim usado mesmo em uma aplicação em camadas.

O comando SQL a ser executado no ClientDataset quebraria um pouco a regra de não se executar comandos SQL (que tem muito mais relação com regras de negócio) na camada cliente. Numa aplicação multicamadas os comandos SQL seriam todos executados no servidor DataSnap.

A vantagem que se obtém de se poder montar um SQL dinamicamente e executá-lo a partir de um ClientDataset é montar um SQL inteligente, que traga poucos dados, diminuindo o tráfego de informações na rede e otimizando a velocidade. Outra vantagem é que se o comando for o mesmo, simplesmente não se faz a consulta, mas aproveita-se os dados que já estão armazenados no CDS.

**Nota**

As versões mais antigas do Delphi não possuem o componente TButonedEdit, mas pode ser usado o TJvButtonEdit da biblioteca JEDI, que analogamente ao TLabeledEdit da paleta Additional ele já vem com um botão. Este é interessante porque o botão tem o mesmo aspecto de um botão comum, e é mais parecido, visualmente, com um combo. A desvantagem do TJvButtonEdit em relação ao TButtonedEdit é que o TButtonedEdit pode ter dois botões, um à esquerda e um à direita.

Esta classe ainda teria a missão de criar dinamicamente um formulário, um datasource associado ao CDS externo e um dbGrid ligado a esse datasource. O próximo passo é fazer com que essa classe descenda de um TButtonedEdit em vez de ser uma classe comum.

Todos os componentes da VCL são descendentes de um TCustom<NomeDoComponente>. Por exemplo o TEdit (e outros parecidos) é descendente do TCustomEdit.

Os componentes prefixados de TCustom mantêm a maioria das propriedades e métodos como protected em vez de public, assim os descendentes desses componentes podem escolher quais membros serão públicos e quais serão privados. É interessante esconder detalhes de implementação que não podem ser alterados por outros programadores.

## Requisitos de um componente

Um componente é uma classe especial que é criada em tempo de design pelo IDE do Delphi. Muito código pode ser executado durante a criação de um programa.

Para a criação de um componente profissional são necessárias classes especiais chamadas property editors e component editors.

Property editors são as classes que ajudarão o Object Inspector a descobrir e exibir corretamente uma propriedade, ou facilitar a edição dela.

Por exemplo os componentes data-aware, como os lookups, não requerem que o programador digite o nome do campo associado, mas quando corretamente conectados a um datasource e dataset a propriedade DataField exibe uma lista de campos que o programador pode escolher.

Component editors são classes que agem ao se clicar no próprio componente, e que auxiliam sua configuração. O editor de fields de um dataset ou o editor de colunas de um dbgrid são exemplos de component editors.

Além de editores de propriedades e de componente, um componente profissional precisa de imagens, por exemplo uma "seta" a ser clicada. Essas imagens podem ser embutidas em um arquivo .dcr, que é um arquivo de resources (arquivo de recursos) próprio para componentes.

Todo componente deve ser registrado no IDE para aparecer na paleta. Para isso é necessário criar a procedure Register começada com R maiúsculo. Essa procedure registrará o componente no IDE chamando o método RegisterComponents(Nome,[componentes]). O primeiro argumento é o nome da paleta ou seção onde os componentes aparecerão. Neste artigo o nome da paleta escolhido é CD, de ClubeDelphi. O segundo argumento é um vetor de componentes.

Pode ser registrado mais de um componente de uma vez, caso a unit tenha vários componentes ou se execute a procedure Register em uma outra unit que faça apenas isso: registrar os componentes contidos nas outras units.

Para que o componente tenha um ícone que possa ser mostrado no IDE é necessário criar um arquivo .DCR. Arquivos DCR são como arquivos RES, de resources, próprios para componentes. O Image Editor que vinha no Delphi 7, e em outras versões anteriores, pode ser usado para criar o arquivo DCR. Embora o Delphi XE não venha mais com um editor de imagens pode ser usado um editor de terceiros para criar arquivos DCR ou pode ser usado o brcc32.exe. Na seção de links há páginas que mostram as duas maneiras de se fazer os arquivos dcr.

O arquivo DCR deve ter o mesmo nome que a unit onde estão os componentes. Caso contrário deve ser incluída no projeto com a diretiva {$R NomeDoArquivo.dcr} logo na seção implementation da unit do componente.

Dentro do arquivo deve haver um bitmap com 16x16 ou 22x22 pixels com o mesmíssimo nome da classe do componente(TCDLOOKUP) porém tudo em letras maiúsculas. Caso o nome seja diferente ou não esteja em letras maiúsculas o componente não terá imagem própria no IDE do Delphi.

Outros arquivos de imagens podem ser incorporados dentro desse DCR, como é o caso de SETA.BMP, que é um arquivo de uma seta vermelha para baixo que foi usado nesse componente como imagem para o RightButton. Por convenção o componente criado neste artigo se chamará TCDLookup.

Como o TCDLookup criado é descendente de TButtonedEdit, algumas vezes as expressões Text ou Edit citadas neste texto se referirão a essa porção TButtonedEdit de TCDLookup. Lembrando que TButtonedEdit é filho de TCustomEdit, o que o torna parente de TEdits comuns.

## Preparando o ambiente

Não adianta fazer um lookup especial sem ter onde testá-lo. Nesse artigo foi escolhido o banco de dados de exemplo employee.fdb que vem no diretório de instalação do Firebird. A tabela mestre escolhida foi a de departamentos (DEPARTMENT) e a tabela detalhe foi a de empregados (EMPLOYEE). Um empregado possui um departamento, mas um departamento possui n empregados. A arquitetura escolhida para acesso ao banco foi o dbxpress: TSQLDataset + Provider + ClientDataset. A versão do Delphi escolhida foi a XE e a versão do Firebird escolhida foi a 2.5.

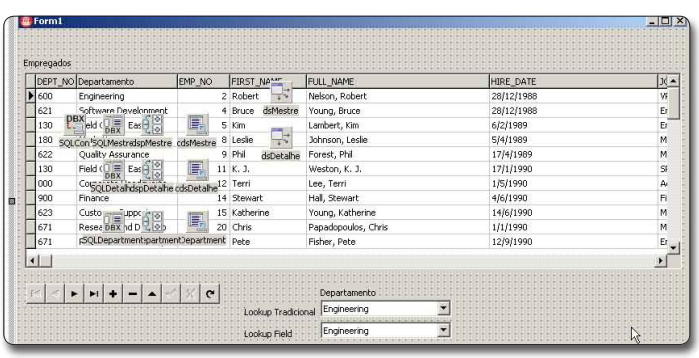



**Figura 1.** Dois componentes lookup, um tradicional e outro "arrastado" a partir de um LookupField

Para se estabelecer uma comparação com lookups tradicionais foi criado um formulário apresentando o uso desses controles, que pode ser obtido juntamente com o source desse artigo, lá podem ser vistos os comandos SQL usados nos dois TSQLDataSet.

A **Figura 1** apresenta um formulário com componentes de lookup tradicionais e provindos de campos lookup. A **Figura 2** mostra os property editors desses componentes.

Um lookup deve ter cinco propriedades principais:

DataSource → assim como os outros controles conscientes de dados, como os TDbEdit, este é um componente do tipo TDataSource ligado a um TDataset;

DataField → analogamente ao TEdit é o campo que será editado. No caso de um lookup, é a chave estrangeira DEPT_NO na tabela EMPLOYEE;

ListSource → É o datasource ligado ao dataset ou tabela que mostrará uma lista de registros que quando selecionados fornecerão o valor da chave primária correta a ser armazenada no DataField;

ListField → É o campo ou conjunto de campos da tabela DEPARTMENT que será mostrado ao usuário para que ele reconheça um departamento;

KeyField → É a chave primária de DEPARTMENT.

O lookup montado nesse artigo tem uma peculiaridade especial: em vez de aceitar um DataSource como ListSource ele aceitará um ClientDataset para que possa executar comandos personalizados com a propriedade commandtext. A escolha do ClientDataSet como ListSource permite que qualquer comando ou filtro personalizado seja criado para exibição dos registros da tabela mestre e mesmo assim permite que o componente funcione com qualquer banco de dados, já que o ClientDataset será ligado a um DataSetProvider e este pode estar ligado a qualquer tipo de DataSet, de qualquer banco. Pode ser DBXpress, Zeos, IB, MyDac ou outro de sua preferência. Ele precisará de uma propriedade do tipo string para definir o nome da tabela (ou view) já que o SQL que trará os campos será montado dinamicamente.

É interessante que as cinco propriedades especiais sejam verificadas durante a criação em tempo de execução, disparando uma exceção caso não estejam definidas, mas que não sejam verificadas em tempo de design, caso contrário o simples fato de colocar um componente no formulário dispararia uma exceção.

Como uma comodidade extra esse lookup pode ter filtros adicionais, que seriam cláusulas adicionadas no "where" do SQL. Basta usar TStringList e um pouco de imaginação para montar comandos SQL extremamente precisos de acordo com o contexto.

Para os testes, entretanto, não é necessário um botão para executar ApplyUpdates porque o objetivo não é verificar se o registro alterado está sendo persistido no banco. Tendo ele sido modificado temporariamente pelo "post" e sendo visto modificado no cdsDetalhe já está de bom tamanho.

**Nota**

Como o componente lookup feito nesse artigo altera dinamicamente o commandText do ClientDataSet ligado a ele como ListSource é aconselhável que o CDLookup seja conectado e testado com um ClientDataSet separado, caso contrário os lookups tradicionais ligados ao mesmo ClientDataSet também trarão os resultados filtrados.

## O "mostrador" do lookup

Em vez de se usar canvas puro ou frames pode-se usar um formulário para fazer o "mostrador" ou "aba" do lookup. Esse mostrador servirá para simular um combo, mas feito com um formulário ele pode ser muito mais flexível, podendo inclusive ser arrastado ou expandido pelo usuário.

Esse formulário precisará de tratamento especial ao se clicar em seu grid, ou ao se fechar o formulário, pois ele tem que passar para o componente de lookup o valor do campo chave do registro escolhido ou simplesmente não passar nada caso tenha sido fechado sem que nada fosse selecionado.

A **Listagem 1** mostra como esse mostrador pode ser, mantendo sua funcionalidade mais básica, como já mencionado, os fontes estão disponíveis para download, portanto apenas os métodos do formulário são exibidos. Obviamente que o formulário pode ser alterado e ampliado para atender qualquer necessidade especial do projeto onde será inserido.

Explicando os métodos, na criação do formulário são criados também um DBGrid e um DataSource. Esses são destruídos no destroy, mas apenas depois que as referências a eles sejam removidas.

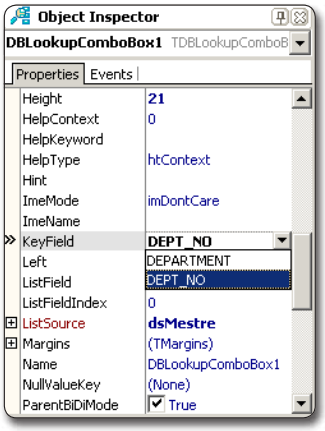



**Figura 2.** Algumas propriedades não precisam ser digitadas graças a um Property Editor

CMExit monitora a mensagem CM_Exit para determinar se o usuário "saiu" do formulário do lookup, tanto por pressionar tab como por clicar em outro lugar e sair do formulário.

DBGExit é um "evento" (método com assinatura de um tipo predeterminado que combina com uma propriedade do tipo referência a método). Esse evento ocorrerá de forma análoga sempre que o usuário abandonar o dbGrid. TabClick manda fechar o formulário executando o método Close caso o dbGrid seja clicado. Se o dbGrid for clicado o cursor do dataset conectado a ele será posicionado em um registro, sendo este o registro escolhido. TabKeyPress monitora se o usuário selecionou um registro com enter ou se ele desistiu com esc. Caso o usuário tenha desistido da seleção a propriedade Abandonou será verdadeira e nada deverá acontecer.

**Listagem 1.** Formulário embutido no lookup

```
procedure TCDLookUpForm.TabKeyPress(sender: tobject; var key: char);
begin
  FAbandonou := false;
  if key in [#13] then
    close;
  if key = #27 then
  begin
    FAbandonou := true;
    close;
  end;
end;
procedure TCDLookUpForm.Close;
begin
  inherited Close;
end;
procedure TCDLookUpForm.DBGExit(Sender: TObject);
begin
  Close;
end;
destructor TCDLookUpForm.Destroy;
begin
  FDataSource.DataSet := nil;
  FGridConsulta.DataSource := nil;
  FDataSource.Free;
  FGridConsulta.Free;
  inherited;
end;
procedure TCDLookUpForm.keypress(var key: char);
begin
  inherited KeyPress(key);
end;
procedure TCDLookUpForm.tabclick(sender: tcolumn);
begin
  FAbandonou := false;
  Close;
end;
constructor TCDLookUpForm.CreateNew(AOwner: TComponent; Dummy: Integer = 0);
begin
  inherited CreateNew(aowner, Dummy);
  BorderIcons := [biSystemMenu];
  if not assigned(FGridConsulta) then
  begin
    FDataSource := TDataSource.Create(nil);
    FGridConsulta := Tdbgrid.Create(nil);
    FGridConsulta.Name := 'FGridConsulta';
    FGridConsulta.Options := FGridConsulta.Options - [dgEditing, dgIndicator, dgTabs];
    FGridConsulta.Parent := self;
    FGridConsulta.ReadOnly := true;
    FGridConsulta.Align := alClient;
    FGridConsulta.DataSource := FDataSource;
    //eventos
    self.FGridConsulta.OnCellClick := tabclick;
    self.Caption := '';
    FGridConsulta.OnKeyPress := TabKeyPress;
    FGridConsulta.OnExit := DBGExit;
  end;
end;
procedure TCDLookUpForm.CMExit(var Message: TCMExit);
begin
  inherited;
  Close;
end;
```

## Partes básicas do componente

Um componente de acesso a dados deve ter internamente um objeto da classe TFieldDataLink. Este objeto será responsável por fazer a conexão entre o componente e o DataSource. É dele que virá a informação de qual DataField está sendo observado e é ele que atualizará o que aparece escrito no componente.

Alguns métodos obrigatoriamente devem ser criados para que cada vez que se muda o registro selecionado no lookup ele mude o CDS Detalhe (Employee) ou cada vez que se navegue no detalhe ele automaticamente mude o nome do departamento apresentado no lookup.

Esses métodos especiais são eventos do FDataLink, o objeto interno do tipo TFieldDataLink. São eles: DataChange, EditingChange e UpdateData.

DataChange verifica se o KeyValue foi modificado, talvez por se navegar, incluir ou reabrir o CDSDetalhe. EditingChange faz com que o TButtonedEdit fique somente leitura se o DataLink não estiver em modo de edição. UpdateData faz o caminho inverso de DataChange e muda a chave estrangeira no CDSDetalhe quando um registro diferente é escolhido no lookup.

Como o código fonte é muito extenso a seção interface da unit e a definição da classe não são incluídas nas listagens desse artigo, mas devem ser conferidas no source dessa edição. Na definição da classe há muitos fields privados que definem valores de propriedades ou são usados em operações pelos métodos. Na **Listagem 2** podem ser conferidos os métodos DataChange, EditingChange e UpdateData.

O DataChange mudará o texto apresentado e o KeyValue do componente de acordo com o registro apontado no CDSDetalhe, ou seja, de acordo com o valor de FDataLink.Field.Value. Esse texto é buscado com auxílio do método GetFieldValue que será explicado mais adiante.

Se ligações essenciais como ListSource ou ListField não estiverem definidas o DataChange não é executado até o final. Ele também define o tamanho máximo do TButtonedEdit (MaxLength) de acordo com o tamanho do campo a ser editado, e somente em tempo de execução. Em tempo de design o DataChange garante que o componente terá o Text igual ao próprio nome caso esteja desligado de todos os datasources e datasets.

O EditingChange define que o TButtonedEdit será ReadOnly sempre que ele não estiver em edição. O UpdateData finalmente grava no FDataLink.Field.Value o valor da chave selecionada, consequentemente gravando esse valor na tabela detalhe da mesma forma que um lookup faz.

Outros métodos importantes são os métodos que recuperam o valor corrente apontado pelo lookup, chamado GetFieldValue, o método que abrirá o formulário, chamado Execute e SetSQL que cria a sentença SQL. Ambos podem ser encontrados no código para download.O método GetFieldValue só é executado até o final quando o componente (e o datasource dele) está em modo de navegação. Se ele entra em modo de inserção ou edição ele sairá do método antes de qualquer coisa. A responsabilidade desse método é, em modo navegação, dado um KeyValue qualquer vindo da tabela em que se está navegando, consultar na tabela mestre uma consulta para trazer o campo dessa tabela que será mostrado. Lembrando que campos lookup lidam com um campo, editando-o, mas mostram outro, da tabela de origem, justamente para que o usuário não tenha que ver ou saber os valores das chaves primárias e estrangeiras. No exemplo desse artigo o campo TCDLookup serve para editar o campo DEPT_NO na tabela employee, mas sempre irá mostrar o campo Department da tabela Department, tanto em modo de edição como em modo de navegação.

O método SetSql atribui à variável FSQL a instrução mais básica de todas para ser executada no CommandText do ClientDataSet que for a ListSource do componente. Geralmente será um "select chave, campo from tabela" sem where. É este método e esta variável que devem ser alterados para se colocar filtros adicionais no where ou campos adicionais para se exibir no DbGrid. Por exemplo o programador, ao alterar o componente, pode adicionar duas propriedades no componente, CampoAdicional e FiltroAdicional, e preencher as propriedades, de maneira que o SQL poderia

**Nota**

Eventos nos componentes do Delphi são na verdade propriedades de um tipo especial de dado que é uma referência a um método. Assim como métodos, os eventos têm assinatura e apenas casam com métodos de assinatura semelhante. Na **Listagem 1** o evento OnExit do DBGrid recebe o método DBGExit da seguinte forma: FGridConsulta.OnExit := DBGExit.

### Nota do DevMan

Diferentemente de aplicações console ou em lote que executam todas as suas funções em uma linha vertical e depois terminam, ou de uma aplicação Web que roda em um servidor Web, espera por uma requisição, processa e responde apropriadamente, uma aplicação Windows se comporta de maneira diferente.

Uma aplicação Windows roda em uma "janela",e toda janela é na verdade um loop que só termina quando se clica no fechar (X).Toda a aplicação é executada dentro desse loop, e para que a aplicação não fique travada e permita interação do mouse e teclado com os controles, botões e edits, a aplicação a cada "laço" recebe e processa mensagens da API do Windows.

Estas mensagens que levam para a aplicação as teclas pressionadas em um teclado, os movimentos do mouse e outros periféricos e até mesmo mensagens de outras janelas, programas e componentes. No componente desenvolvido nesse artigo é monitorada a mensagem CM_EXIT, criada especialmente para a VCL, na unit Controls. CM_Exit tem o valor CM_BASE, que é $B000, adicionado de 27, resultando em $B01B, ou 45083 em decimal. Essa mensagem é disparada sempre que se sai (ou se abandona) um componente visual por se clicar em outro ou pressionar TAB. Quando isso ocorre o valor 45083 é passado para a função que monitora as mensagens, processando-a de acordo com o tipo de mensagem, e o que estiver na procedure que monitora CM_Exit na assinatura será executado, inclusive o evento OnExit, dentro de DoExit no caso do Edit do Lookup, ou simplesmente fecha a janela no caso do formulário que atua como "display" do lookup.

A unit Windows e a unit Messages têm vários outros exemplos de mensagens, e para monitorar, interceptar e enviar mensagens elas fazem uso massivo da API do Windows.

O programador experiente pode notar que todas as procedures ou functions que tratam mensagens têm a diretiva "message" seguida de uma mensagem do sistema logo após sua declaração. O compilador do Delphi trata essas funções de modo especial, sendo que elas são invocadas mesmo que o fluxo de execução do programa esteja em algum outro controle, evento ou loop. O programador não precisa se preocupar nesse caso com threads, linhas de execução, loops, interrupções e todos esses assuntos de baixo nível.

Se dentro de um evento, como o clique de um botão, o programador colocar um loop independente de ele ser simplesmente longo ou infinito, o programa irá congelar durante a execução do loop, não permitindo cliques em outros controles. Para que isso não aconteça e que o programa permita a interação com outros controles e até mesmo o cancelamento dentro de um loop, ao final do mesmo, antes do "end" deve ser executado o método Application.ProcessMessages. Esse método irá desenfileirar, receber e executar as mensagens mandadas a outros controles na janela, mesmo durante um loop.

**Listagem 2**. Métodos que monitoram alterações nos dados de FDataLink

```
procedure TCDLookup.DataChange(Sender: TObject);
begin
 //se ainda não foi definido o CDS fonte sair
 if FListSource = nil then
  exit;
 if FDataLink.Field <> nil then
 begin
  FKeyValue := FDatalink.Field.Value;
  Text := GetFieldValue;
  if (FListSource.FindField(FListField) <> nil) and
   (Alignment <> FListSource.Fieldbyname(FListField).Alignment) then
  begin
   Alignment := FListSource.Fieldbyname(FListField).Alignment; //força atualizacao do
   alinhamento
  end;
  if not (csDesigning in ComponentState) then
  begin
   //em modo de edição, apenas em runtime e sendo o campo
   //do tipo texto define o MaxLength do TButtonedEdit como
   //sendo o tamanho do campo buscado
   if (FListSource.FieldByName(FListField).DataType in
    [ftString, ftWideString]) and (MaxLength = 0) then
     MaxLength := FListSource.FieldByName(FListField).Size;
  end;
  if Focused and FDataLink.CanModify then
  else
  begin
   if (FDataLink.Editing and FDataLink.DataSet.Modified) then
    Modified := True;
  end;
 end
 else
 begin
  Alignment := taLeftJustify;
  if csDesigning in ComponentState then //text = nome somente em tempo de design
   Text := Name
  else
   Text := '';
 end;end;
procedure TCDLookup.EditingChange(Sender: TObject);
begin
 if ASSIGNED(FDataLink.DataSource) then
  inherited ReadOnly := not FDataLink.Editing;
end;
procedure TCDLookup.UpdateData(Sender: TObject);
begin
 if Assigned(FDatalink) and
  Assigned(FDatalink.DataSource) and
  Assigned(FDatalink.DataSource.DataSet) and
  FDatalink.DataSource.DataSet.active then
  begin
   FDataLink.Field.Value := FKeyValue;
  end;
end;
```

ficar: "select chave, campo, CampoAdicional from tabela where FiltroAdicional". Dessa maneira mais campos podem ser exibidos e menos registros trazidos do banco.

O método Execute é o que concentra a maior parte do funcionamento desse componente. Ele é responsável por fechar o ListSource, definir uma instrução SQL e reabrir o ListSource. A instrução SQL pode ter um filtro do tipo Like caso algumas letras sejam digitadas no Edit ou pode executar um SQL básico sem where que traz todos os registros.

**Nota**

Embora o objetivo desse componente seja executar instruções SQL mais enxutas e trazer menos registros, e atualizá-los, isso pode ser uma faca de dois gumes. Sempre que se navegar pelos registros uma instrução SQL será executada para obtenção do nome ou campo associado a essa chave primária, e sempre que um SQL básico for montado, sem um filtro where e sem que algumas letras tenham sido digitadas no TButtonedEdit uma instrução que traz todos os registros será executada. Há duas maneiras de se resolver isso: uma delas é colocar internamente no componente um segundo ClientDataSet onde tudo que for trazido do banco de dados será copiado para ele para que possa servir de buffer. Outra maneira é deixar o componente não data-aware e não permitir que ele faça consultas sem nada digitado, ou quando estiver em modo de navegação. Fará consultas apenas em modo de edição. Isso traria a desvantagem de não mostrar o texto para o registro corrente em modo de navegação, mas uma combinação desta técnica com o buffer poderia resolver esse problema.

Depois de definida a instrução SQL o método Execute instancia o formulário que é o "display" do lookup, cria um ponto PT dando a ele as coordenadas (X,Y) logo abaixo do próprio componente e usando pt := Self.ClientToScreen(pt) para tirar a diferença entre o local do formulário principal e o local onde o componente "começa". Assim é definida a posição e estilo de borda do formulário e logo depois disso ele é exibido com ShowModal.

Depois que ele é fechado, caso o usuário tenha desistido da edição pressionando ESC o formulário fechará sem que nada aconteça. Caso o usuário tenha clicado ou pressionado Enter em um registro qualquer, esse registro é o que aparecerá escrito em Text e a chave primária, ou KeyValue, será atribuída a FDataLing. Field.Value, ou seja, ao registro corrente do dataset ligado ao componente, modificando-o.

Antes de se destruir o formulário de display são executados os métodos FDataLink.Modified, FDataLink.UpdateRecord, UpdateData(self) para que o registro seja apropriadamente atualizado e mostrado. DoExit é executado para que qualquer evento OnExit no componente seja disparado e SetFocused(false) é executado para que o componente não ganhe novamente o foco.

Além desses métodos são necessários métodos importantíssimos para monitorar eventos e colocar o DataLink automaticamente em modo de edição ou navegação caso se realize alguma digitação, copiar, colar ou recortar no TButtonedEdit. Esses são métodos que monitoram mensagens da API do Windows e que a maioria dos componentes visuais implementa:

- WMCut – Acontecerá sempre que o usuário recortar o conteúdo do TButtonedEdit. Requer um Edit no FDataLink;
- WMPaste – Sempre que o usuário colar algo no controle. Requer um Edit no FDataLink;
- WMUndo – Sempre que o usuário pressionar ctrl+z e desfazer a última alteração. Requer um Edit no FDataLink;
- CMEnter – Sempre que o usuário entrar no componente;

- CMExit – Sempre que o usuário deixar o componente;
- WMKeyDown – Sempre que o usuário pressionar uma tecla.

Além desses, o método DoEnter precisa ser sobrecarregado para que cada vez que alguma operação for feita no componente, o Texto atual seja reservado fazendo-se uma cópia para FTextOld. O Change precisa ser sobrecarregado para que qualquer mudança no Edit chame o método FDataLink.Modified do DataLink, para que de fato seja registrada uma mudança e ele entre em modo de edição. A **Listagem 3** mostra como é feito o tratamento das mensagens do Windows.

Os métodos WMPaste, WMCut, WMUndo simplesmente monitoram as mensagens colar, recortar e desfazer do windows, colocando o dataset do lookup em modo de edição.

CMEnter coloca o foco e deixa o controle editável retirando a propriedade readonly quando o campo puder ser editado. CMExit faz o caminho inverso, executando o evento OnExit quando apropriado e tomando o cuidado de setar o KeyValue para null se o text for string vazia.

WMKeyDown verifica se a tecla pressionada é a tecla definida como NullValueKey. Se for então o campo será editado e será atribuído o valor null. Para que isso funcione é necessário a presença da unit menus no uses, para que se aproveite a definição do método Shortcut.

**Listagem 3**. Métodos que tratam mensagens da API do Windows

```
procedure TCDLookup.CMEnter(var Message: TCMEnter);
begin
 SetFocused(True);
 inherited;
 if ASSIGNED(FDataLink.DataSource)  and FDataLink.CanModify then
   inherited ReadOnly := False;
end;
procedure TCDLookup.CMExit(var Message: TCMExit);
begin
 SetFocused(False);
 if trim(Text) = '' then
  FKeyValue := null;
 DoExit;
 inherited;
end;
procedure TCDLookup.WMCut(var Message: TMessage);
begin
 FDataLink.Edit;
 inherited;
end;
procedure TCDLookup.WMKeyDown(var Message: TWMKeyDown);
begin
 if (FNullValueKey <> 0) and (FNullValueKey = ShortCut(Message.CharCode,
  KeyDataToShiftState(Message.KeyData))) then
 begin
  if FDataLink.CanModify then
    FDataLink.DataSet.Edit;
  SetKeyValue(null);
  Message.CharCode := 0;
 end;
 inherited;
end;
procedure TCDLookup.WMPaste(var Message: TMessage);
begin
 FDataLink.Edit;
 inherited;
end;
procedure TCDLookup.WMUndo(var Message: TMessage);
begin
 FDataLink.Edit;
 inherited;
end;
```

É necessário que o TButtonedEdit tenha um ImageList associado. Esse ImageList é criado (e destruído) internamente, e as imagens necessárias são adicionadas dinamicamente na hora da criação. Essas imagens podem ser carregadas de arquivos BMP ou de resources. Caso se carregue de um arquivo BMP o caminho pode ser transformado em uma propriedade.

O botão será perceptível somente se houver uma imagem. Nesse artigo foi usada uma seta vermelha para baixo que tradicionalmente vem junto com o Delphi, na pasta C:\Arquivos de programas\Arquivos comuns\CodeGear Shared\Images\Buttons.

O componente TCDLookup terá internamente forms, dbgrids, datasource e referências a ClientDataSet. Para que possa ter acesso a tudo isso as seguintes units serão necessárias no uses: SysUtils, forms, windows, dbctrls, Controls, StdCtrls, classes, DBGrids, db, dbclient, ExtCtrls, variants, graphics, messages e menus.

Nesse ponto o componente já pode ser compilado, instalado e colocado em um formulário para testar. Não se deve esquecer de preencher a propriedade TableName no object inspector, senão o componente não saberá o nome da tabela na hora de montar a instrução SQL para rodar no CommandText.

A **Figura 3** mostra o componente trocando o departamento de um empregado e a **Figura 4** mostra o componente filtrando por Custom.

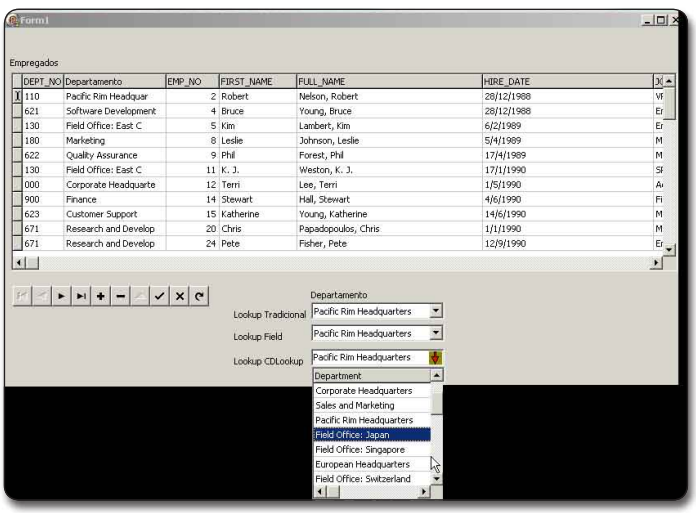



**Figura 3.** O TCDLookup em execução, modificando um registro

É possível verificar que filtrando por Custom apenas dois registros são trazidos. No método Execute citado anteriormente o comando SQL é montado de forma que o filtro seja sensível à caixa. Para que seja case insensitive deve ser usado o seguinte SQL:

```
FSQL := 'select ' + FKeyField + ',' + FListField + ' from ' + FTableName + ' where lower(' +
FListField + ') like ' + QuotedStr('%'+LowerCase(Text)+'%');
```

## Eventos

Adicionar um evento é fácil. Basta criar uma variável privada do tipo TNotifyEvent e uma propriedade que acesse esse Field na

seção published. Automaticamente um evento novo aparecerá no object inspector. Como exemplo pode ser criado o evento BeforeShowForm que pode ser definido como na código a seguir:

```
private
 FBeforeShowForm: TNotifyEvent;
Published
 property BeforeShowForm: TNotifyEvent read FBeforeShowForm write FBeforeShowForm;
```

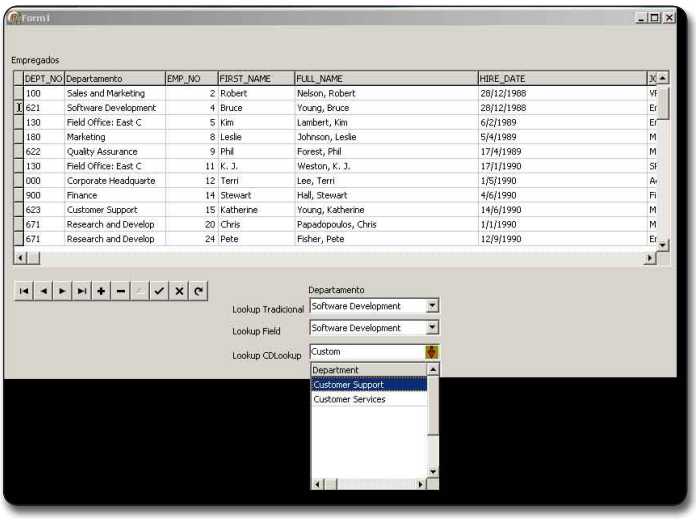



**Figura 4.** O TCDLookup em execução, filtrando por Custom e trazendo apenas 2 registros

Para executar o evento basta ir no ponto apropriado do código, verificar se o evento existe e é diferente de nil, já que o programador pode simplesmente deixar de implementar esse evento, e caso exista executá-lo chamando FBeforeShowForm como se fosse um método.

Se o programador definir um evento para BeforeShowForm dando nele um duplo clique no object inspector então o evento existirá e será executado. Caso contrário o evento será nil e não será executado. O seguinte código mostra onde executar o evento e como protegê-lo caso ele não esteja definido. Se essa verificação não fosse feita seria disparado um Access Violation:

```
procedure TCDLookup.RightButtonClick(Sender: TObject);
begin
 //se houver um evento a ocorrer após o clique do botão seria aqui
 if Assigned(FBeforeShowForm) then
  FBeforeShowForm(self);
 Execute;
end;
```

A **Figura 5** mostra como o evento novo aparece no Object Inspector.

## Property Editors

Sem property editors o componente TCDLookup terá apenas as propriedades DataSource, DataField e ListSource automaticamente editáveis. Isso porque nos casos do DataSource e do ListSource o mecanismo da VCL e da RTL por trás do IDE consegue verificar que tratam-se de elos com componentes que podem estar presentes na mesma form, por isso os componentes da mesma classe das propriedades são exibidos para que possam ser escolhidos.

### Nota do DevMan

Depurar um componente é uma tarefa árdua: Algumas vezes será necessário colocar o próprio Delphi (bds.exe) como host application, ou abrir uma outra instância do Delphi e dar o comando Run – Attach to Proccess para usar o próprio Delphi para depurar aspectos do comportamento do componente em Desgin Time, pois é no IDE que os editores de propriedades e de componente funcionam. Além disso o construtor também é executado no IDE. Outras vezes será necessário criar um programa de exemplo trivial apenas para testar o componente, e ele deverá ser host application da biblioteca/pacote em questão, para que outros aspectos em tempo de execução possam ser depurados. O ideal é ter essa aplicação de exemplo no mesmo Project group do pacote do componente, para facilitar a depuração. Sempre que for alterada alguma coisa no source do componente é altamente recomendado dar os comandos Uninstall e Clean (clicando com o botão direito no Project manager) para que todas as bpls e dcus envolvidos sejam deletados e uma nova compilação completa seja feita com o comando Install, para se fazer novos testes. Nas versões antigas do Delphi era necessário reinicializar a IDE a cada desinstalação/reinstalação do componente sendo desenvolvido.

**Nota**

Grande parte dos métodos desse componente e rotinas para atualizar o alinhamento correto foram simplesmente copiados do TDBEdit da VCL. O TDBEdit é descendente de TCustomEdit e é um ótimo exemplo de como se transforma um componente "comum" em um componente "Data-Aware".

**Nota**

TNotifyEvent na verdade é uma referência para método de objeto, e sua definição é: TNotifyEvent = procedure(Sender: TObject) of object;

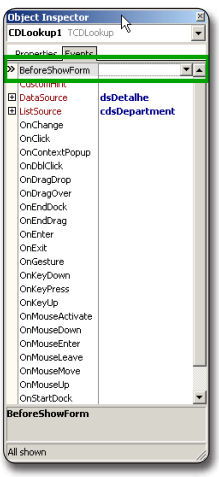



**Figura 5.** Evento BeforeShowForm

A propriedade DataFiled pode ser editável automaticamente por estar ligada a um DataLink e por se chamar "DataField". Com esse nome o IDE usa RTTI para vasculhar no DataSource, depois no DataSet e listar os nomes de campos que encontrar ali. Por isso que não dá para escolher um DataField antes de se definir um DataSet, ou seja, antes de definir o DataSet a propriedade DataField não é editável. O Delphi já possui uma série de Property Editors padrão para certos tipos ou estruturas de dados, ou dados com significados especiais.

Por enquanto, se instalado dessa maneira as propriedades ListField e KeyField devem ser digitadas manualmente pelo programador e nada impede de digitar um campo que não existe. Este artigo apresentará os Property Editors para os campos ListField e KeyField, que virão de dentro do ListSource.

A propriedade TableName não terá property editor, mas ela deve ser preenchida sempre. Para assegurar isso no construtor do componente pode haver uma verificação se a propriedade TableName é diferente de '' quando o componente não estiver em tempo de design. Caso TableName seja '' então uma exceção deve ser disparada.

Para criar um property editor é necessário incluir o pacote DesignIDE.dcp na lista (ou pasta) Requires no Project Manager. Esse pacote é o que contém a unit DesignEditors compilada. Essa unit é necessária pois contém as classes e interfaces bases para qualquer property editor que será apresentado no object inspector.

É uma boa prática colocar os property editors em units separadas dos componentes. Melhor ainda é a prática de colocar os property editors em pacotes diferentes do componente. Isso se dá por dois motivos: primeiro porque o property editor vai usar a unit do componente, mas o contrário nunca deve ser verdadeiro. Apenas o Delphi vai usar o property editor. Outro motivo é que o pacote do property editor é importante apenas para programadores, então ele pode ser definido como um pacote de Design Time apenas. O pacote do componente deve ser do tipo Run Time e Design Time.

O property editor desse componente, para as propriedades ListField e KeyField é bem simples de se fazer. Para isso o código foi colocado em uma unit chamada uCDLookUpPED.pas (PED de Property Editor) e as units uCDLookUp, DesignEditors, DesignIntf, classes e DBClient foram acrescentadas no uses. uCDLookUp é necessária pois o property editor necessita "conhecer" o componente para o qual trabalha. DesignEditors é a unit que tem os templates ou classes básicas para todos os editores de propriedade e de componente. DesignIntf é complementar a DesignEditors e possui todas as interfaces para lidar componentes sendo editados, sua relação com o Object Inspector e o trabalho com editores de propriedade. Na maioria dos projetos que se usar DesignEditors se usará DesignIntf também. O método RegisterPropertyEditor está em DesignIntf.

A unit classes foi usada para que se aproveite o TStringList para coletar a lista de campos do ClientDataset ListSource. A unit DBClient foi inserida para que o property editor tenha conhecimento do ClientDataset.

Foi criada uma classe TCDLookupFieldProperty descendente direta de TStringProperty. TStringProperty é a classe padrão para lidar com propriedades de tipo string dos componentes. Para propriedades que também são strings mas que se comportam de maneira diferente, são gerados descendentes de TStringProperty, como no caso de Fields ligados a DataSources ou ainda no caso especial desse artigo, fields ligados diretamente a DataSets sem passar por DataSources.

TStringProperty é descendente de TPropertyEditor. Os dois métodos sobrecarregados em TCDLookupFieldProperty, vistos na **Listagem 4**, são na verdade métodos de TPropertyEditor e não de TStringProperty. Todo o restante do comportamento de TCDLookupFieldProperty será igual ao TStringProperty sem que seja necessário alterar nenhum outro método. Apenas os métodos que definem como a propriedade é apresentada e como a lista de opções é populada são importantes nesse contexto.

O método GetAttributes apenas retorna um conjunto de TPropertyAttributes sem aproveitar nada do método ancestral. O conjunto retornado, [paValueList, paSortList], diz ao IDE que o retorno é uma lista de valores e ordenada.

Os valores possíveis aqui são todos os TPropertyAttribute, ou seja: paValueList, paSubProperties, paDialog, paMultiSelect, paAutoUpdate, paSortList, paReadOnly, paRevertable, paFullWidthName, paVolatileSubProperties, paVCL, paNotNestable, paDisplayReadOnly, paCustomDropDown, paValueEditable.

Os nomes são auto explicativos. Esses atributos se referem ao comportamento da edição da propriedade, e não ao editor ou à propriedade em si.

O método GetValues cria uma StringList, verifica se o ListSource do componente que está sendo editado foi definido, usando if Assigned((GetComponent(0) as TCDLookup).ListSource) e por fim usa (GetComponent(0) as TCDLookup).ListSource. GetFieldNames(sList) para obter a lista de campos do ClientDataset. Depois de obtida a lista ela é varrida e cada valor é executado como parâmetro de proc(sList[i]).

Proc é o parâmetro do método GetValues que é passado pelo próprio IDE. Esse parâmetro é uma referência ou ponteiro para método. Ou seja, o IDE tem um método para coletar os valores que serão apresentados em uma lista de possíveis valores de propriedades. O IDE se encarrega de passar uma referência para esse método ao GetValues do property editor, e o mesmo deve executar esse método passado preenchendo as propriedades. Esse parâmetro Proc é do tipo TGetStrProc que na verdade é um procedure(const S: string) of object.

Se além de adicionar os campos do ClientDataset forem estaticamente adicionadas strings arbitrárias, como 'laranja' e 'abacaxi' essas duas frutas aparecerão como se fossem nomes de campos. Logicamente elas causarão erros se forem escolhidas.

A procedure Register dos property editors é análoga à procedure Register dos componentes. A diferença é que ela registra editores de propriedades e não componentes. Seus argumentos são:

RegisterPropertyEditor(TypeInfo(<tipo da propriedade>), <classe do componente onde a propriedade atua>,< 'Nome da propriedade'>, <Nome do property editor que será registrado para essa propriedade>).

Cada chamada a RegisterPropertyEditor registra um property editor para uma propriedade de um componente com um determinado nome.

Alguns parâmetros podem ser omitidos. Por exemplo, omitindo-se o nome da propriedade e o nome do componente esse property editor será registrado para todas as propriedades do tipo definido no primeiro parâmetro, independente do componente. Só a criação de property editors poderia ter um artigo próprio, pois é uma área bastante extensa, mas não seria interessante deixar o TCDLookup sem um editor de propriedades, mesmo que básico.

A **Figura 6** mostra o property editor em ação, a **Figura 7** mostra que é impossível definir o ListField e KeyField caso não seja definido o ListSource, para isso foi removido o ListSource do componente no formulário de exemplo.

O código fonte desse artigo acompanha um formulário de teste do componente desenvolvido para que possa ser testado tanto em

**Listagem 4**. Property Editors

```
unit uCDLookUpPED;
interface
uses
 uCDLookUp,
 DesignEditors,
 DesignIntf,
 classes,
 DBClient;
type
 TCDLookupFieldProperty = class(TStringProperty)
 public
  function GetAttributes: TPropertyAttributes; override;
  procedure GetValues(proc: TGetStrProc); override;
 end;
procedure Register;
implementation
procedure Register;
begin
 RegisterPropertyEditor(TypeInfo(string), TCDLookup, 'ListField', TCDLookup
 FieldProperty);
 RegisterPropertyEditor(TypeInfo(string), TCDLookup, 'KeyField', TCDLookup
 FieldProperty);
end;
{ TListFieldProperty }
function TCDLookupFieldProperty.GetAttributes: TPropertyAttributes;
begin
 Result := [paValueList, paSortList];
end;
procedure TCDLookupFieldProperty.GetValues(proc: TGetStrProc);
var
 sList: TStringList;
 i: integer;
begin
 inherited;
 sList := TstringList.Create;
 sList.Clear;
 if Assigned((GetComponent(0) as TCDLookup).ListSource) then
 begin
  (GetComponent(0) as TCDLookup).ListSource.GetFieldNames(sList);
  for i := 0 to sList.Count - 1 do
   proc(sList[i]);
 end;
end;
end.
```

runtime como em designtime e para que alterações no componente possam ser acompanhadas quase que instantaneamente. O banco de dados utilizado vem junto com o Firebird 2.5, mas pode ser usado o banco Northwind de exemplo do MSSQL ou qualquer outro banco de exemplo de qualquer outro fabricante.

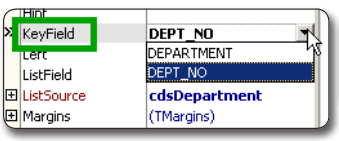



**Figura 6.** Editando KeyField

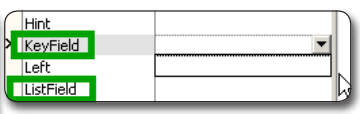



**Figura 7.** Sem ListSource não há sugestões de KeyField ou ListField

## Conclusão

A criação de componentes é uma área que requer conhecimentos intermediários a avançados, não apenas de POO mas conhecimentos específicos do ambiente Delphi. Quando se cria componentes se usa POO para gerar recursos que serão usados em RAD, ou seja, programa-se para ferramentas de desenvolvimento. O usuário deixa de ser um cliente de um software qualquer passando a ser outro programador. Isso faz com que bugs fiquem mais evidentes, mais "visíveis" mesmo que possam ser contornados. Uma grande dose de abstração e por vezes paciência se faz necessário.

O lookup criado neste artigo está longe de ser uma solução completa. Faltam facilidades como inclusão de filtros adicionais, título para o mostrador do lookup, teclas de atalho para seleção de um registro (ou desistência de edição).

É esperado que com a leitura deste artigo o leitor tenha um vislumbre de como funcionam os componentes conscientes de dados e como fazer consultas mais inteligentes, que consumam menos recursos do banco de dados e da rede.

Como ListSource neste artigo foi usado um ClientDataset. Seria interessante experimentar usar como ListSource objetos derivados de TList, vetores de objetos e assim por diante. Para isso seria necessário mesclar esses conhecimentos com generics e RTTI. A nova RTTI do Delphi XE permite fazer esse tipo de coisa. Usando a imaginação poderia-se fazer outros componentes, conscientes de objetos, o que facilitaria bastante o desenvolvimento das GUIs.

**Vitor Luiz Rubio**
*vitorrubio@gmail.com*
*http://blog.vitorrubio.com.br*
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP.
Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais.
Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.


**Blogs do autor**
*http://www.vitorrubio.blogspot.com/*
*http://twitter.com/vitorrubio*

**Como criar arquivos DCR**
*http://eugostododelphi.blogspot.com/2010/01/como-fazer-icone-para-seus-componentes.html*

**Resource Editor para criar arquivos DCR**
*http://melander.dk/reseditor*

**Documentação sobre RegisterPropertyEditor na embarcadero**
*http://docwiki.embarcadero.com/VCL/en/DesignIntf.RegisterPropertyEditor*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Delphi Prism

## XML, Web Services, LINQ, Lambda e AD

A XML (Extensible Markup Language) pode ser usada para trafegar informações entre sistemas de forma desconectada ou off-line e controlar arquivos de configurações, algo que é muito comum no .NET Framework. XML é hoje um padrão que tem sido largamente utilizado em diversos tipos de projetos e é fundamental que todo desenvolvedor saiba como ler, editar, pesquisar e gerar documentos XML. Mais cedo ou mais tarde poderá ocorrer a necessidade de utilizá-los em aplicações. XML pode ser utilizada em diferentes linguagens em diferentes plataformas e sistemas operacionais para intercâmbio.

Quem cria aplicativos comerciais, em algum momento, já precisou trabalhar com arquivos textos (TXT). Senão foi a leitura foi a geração de um arquivo para disponibilizar dados para outras aplicações, integrações com programas bancários através de envio e retorno de remessas, ou para guardar informações próprias para o uso semelhante a configurações do aplicativo. Ao utilizar arquivos desse tipo, surgem diversas dificuldades de manipulação, pois para realizar a leitura do mesmo, precisa ser definida qual regra utilizar. Por exemplo, a definição de um caractere delimitador para determinar quando termina um dado e quando inicia outro, ou então trabalhar com posicionamento fixo para os caracteres. Ao possuir este tipo de modelo o código sempre ficará fixo para o tratamento de um determinado arquivo, e se o proprietário desse arquivo mudar o formato, o código precisará ser modificado para dar suporte ao novo modelo. E, além do mais, o que poderá acontecer é detectar a mudança do mesmo em produção o que gerará diversos constrangimentos aos clientes/usuários finais. No artigo veremos como usar XML para resolver problemas como esse.

De forma semelhante à XML, os Web Services podem ser muito úteis, pois possuem acesso facilitado através da internet e permitem prover conexões a banco de dados ou funções específicas de maneira bastante transparente, por ser baseado em padrões abertos. Web Service não traz na sua proposta o conceito de aplicações com interfaces gráficas para usuários, mas sim serviços que são acessados e disponibilizados em uma linguagem universal, XML, e um protocolo aberto, o SOAP. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Cada aplicação poderá ter a sua própria linguagem de programação, que é traduzida para uma linguagem universal, XML.

### Resumo DevMan

***De que trata o artigo:***

As tarefas mais comuns realizadas em qualquer tipo de aplicativo envolvem a manipulação de documentos XML, acesso a dados e pesquisas em objetos. Este é o foco do artigo, tratar de forma prática o uso de XML, Web Services, LINQ, Expressões Lamba e ADO.NET usando o .NET Framework e Delphi Prism.

***Para que serve:***

XML serve como uma linguagem padrão para integração entre sistemas e armazenamento de dados, que podem ser trocados através de Web Services. A LINQ é uma linguagem unificada que serve para manipular dados em diferentes formatos, incluindo XML e objetos em memória. O ADO.NET é o engine padrão de acesso a dados da plataforma .NET.

***Em que situação o tema é útil:***

XML, Web Services e ADO.NET são úteis na construção de aplicações escaláveis, que necessitam realizar intercâmbio de dados entre diferentes plataformas, conectadas através da Internet. Lambda Expressions podem ser usadas em vários cenários para criar códigos mais simples e enxutos.

***Delphi Prism - XML, Web Services, LINQ, Lambda, ADO.NET e muito mais:***

Este artigo é um guia prático que serve como uma ótima referência para a realização de tarefas básicas e cotidianas no desenvolvimento com o Delphi Prism, como a criação e leitura de documentos XML, uso de Web Services para chamada de procedimentos remotos, além de acesso a dados com ADO.NET, nos modelos conectado e desconectado. Além disso, mostra outras importantes tecnologias usadas também no dia a dia, como LINQ, Expressões Regulares e Expressões Lambda. Como vamos ver, este artigo aborda várias técnicas intimamente relacionadas no desenvolvimento com o .NET e Delphi Prism, porém para fins didáticos, exemplos são apresentados de forma independente.

Para a representação e estruturação dos dados nas mensagens recebidas/enviadas é utilizado XML. As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol), também baseado em XML. Os serviços (operações, mensagens, parâmetros, etc.) são descritos usando a linguagem WSDL (Web Services Description Language), que também é um documento XML. Os Web Services podem ser muito úteis, pois podem ser facilmente acessados e

Rodrigo Araujo

podem prover acesso a banco de dados ou funções específicas de maneira bastante transparente.

Já a linguagem LINQ tem como um dos seus principais objetivos facilitar a consulta a dados em coleções, obtidos de fontes das mais diferentes espécies. Estes podem ter sua origem em uma lista de objetos desenvolvida através de listas dinâmicas ou qualquer outra classe derivada. Os dados podem ser oriundos de registros em uma tabela armazenada em um banco de dados SQL Server ou qualquer outro que seja suportado ou, ainda, serem dados de um arquivo XML.

Nesse contexto, no Prism existem várias tecnologias de apoio e recurso da linguagem, tanto para uso em conjunto com LINQ quanto para uso geral (muitos destes recurso mostrados nos exemplos do artigo, incluindo Lambda). É possível inicializar diretamente uma variável com os resultados de uma consulta. Estão presentes no Prism novidades na linguagem como tipos e métodos anônimos, variáveis tipadas conforme a expressão, ou declaração da variável sem definir o tipo, que poderá ser indicado ou deduzido pela expressão que estará iniciando o valor para a variável, além de inicializadores diretos para objetos e coleções.

No caso das Expressões Lambda, dentro do contexto de métodos anônimos e pesquisas, podemos dizer que são abreviações usadas para expressar um método ou uma condição de pesquisa. Elas apresentam uma forma de escrita bastante diferenciada e sua estrutura básica é composta por um parâmetro de destino seguido de uma expressão lógica, onde vários tipos de dados poderão ser utilizados. Já As expressões regulares são utilizadas para criar determinadas regras de validação de dados, que podem ser informados, analisados e verificados se satisfazem os critérios definidos na expressão. O Delphi Prism apresenta suporte nativo a expressões regulares através do framework. As expressões são planejadas e escritas a partir de pequenos blocos de código, cada bloco individual é bastante simples, possuindo uma sequência lógica distinta. Os blocos podem se agrupar de diversas maneiras para atingir um objetivo específico. As expressões regulares podem ser utilizadas para manipular dados de entrada, realizando testes condicionais, observando as regras definidas, para produzir o resultado final desejado ou, então, informar que a análise não foi válida, pois o fluxo de dados estava em um formato incorreto.

E finalmente, o artigo aborda técnicas de acesso a dados com ADO.NET, que constitui uma evolução no acesso a banco de dados relacionais, proporciona benefícios em relação à questões como concorrência, desempenho e facilidade de gerenciamento, entre outros. Um aspecto interessante do modelo conectado do ADO.NET é a flexibilidade e o controle que seu código proporciona. As instruções SQL são descritas explicitamente e poderão ser ajustadas para adquirir o melhor desempenho, pois não precisam passar por nenhuma camada extra, como no caso do uso de frameworks. No entanto, com todo esse poder e controle surgem vários inconvenientes, pois o código de acesso a dados deve explicitamente gerenciar os recursos do banco de dados. O uso indevido como o não fechamento de uma conexão, poderá gerar problemas de gestão de recursos, causando uma perda no desempenho. Além disso, instruções SQL embutidas em strings estão propensas a erros que não serão resolvidos durante a compilação do projeto (algo que LINQ resolve).

As classes DataReader, Command e Connection possuem implementações específicas para cada provedor de dados, mas sua utilização é transparente. Através da classe Connection, é definida uma conexão (e suas operações de abertura e fechamento que ficam a cargo do desenvolvedor), que irá servir como um canal de comunicação e gerenciamento com o BD. Uma vez estabelecida a conexão, ela é utilizada e transmitida para um objeto Command, que fica responsável por informar ao banco de dados uma tarefa a ser realizada (poderá ser qualquer operação CRUD).

## XML no Prism

Através de um arquivo XML inúmeras vantagens são apresentadas, que não estão presentes em outros formatos de arquivos, como TXT. É um modelo de arquivo que segue uma padronização, é multiplataforma e independente de linguagem. Um arquivo XML é apenas um arquivo de texto que é delimitado por TAGS criadas pelo próprio usuário, o documento em si não faz coisa alguma só armazena o conteúdo.

Um documento XML pode ser criado em qualquer editor de textos, depois de criado o arquivo, é escrito código para interpretá-lo. Porém, existem diversas ferramentas para criar um arquivo XML, interpretá-lo, e funções para realizar leitura, escrita e exibição de dados.

Dentre as principais vantagens de se utilizar um arquivo XML pode-se citar a personalização de acordo com a necessidade e delimitação de dados (de forma hierárquica), manutenção facilitada, simplicidade de uso e criação e a portabilidade para diversos ambientes.

Como exemplo, veja a **Listagem 1**, que define a criação de um arquivo XML simples no Delphi Prism, nela são definidos alguns atributos para uma "agenda", contendo o nome da pessoa e seu telefone. Esta é uma maneira tradicional de se criar um arquivo XML. A estrutura do arquivo criado é visualizada na **Listagem 2**.

Uma variação para a criação do arquivo XML poderá ser obtida através do uso da classe XmlTextWriter. Ela contém métodos e propriedades para escrever documentos XML, apresentando diversas formas de escrita através de WriteNode, WriteString, WriteAttributes, WriteStartElement e WriteEndElement, alguns utilizados em conjunto, como o início e o fim de um elemento. Para escrever um elemento deverá ser utilizado WriteStartElement, o valor do seu elemento, caso tenha, e ao seu final WriteEndElement.

**Listagem 1.** Criação de um arquivo XML

```
var xml: System.Xml.XmlDocument := new System.Xml.XmlDocument();
var nome: System.Xml.XmlElement := xml.CreateElement('Nome');
nome.InnerText := 'Rodrigo';
var fone: System.Xml.XmlElement := xml.CreateElement('Fone');
fone.InnerText := '123456789';
var contato: System.Xml.XmlElement := xml.CreateElement('Contato');
contato.AppendChild(nome);
contato.AppendChild(fone);
var agenda: System.Xml.XmlElement := xml.CreateElement('Agenda');
agenda.AppendChild(contato);
xml.AppendChild(agenda);
MessageBox.Show(xml.InnerXml);
```

**Listagem 2.** Estrutura do arquivo XML criado

```
<Agenda>
  <Contato>
    <Nome>Rodrigo</Nome>
    <Fone>123456789</Fone>
  </Contato>
</Agenda>
```

**Listagem 3.** Criação de um arquivo XML utilizando XmlTextWriter

```
var textWriter: XmlTextWriter := new XmlTextWriter('C:\arquivo.xml', nil);
textWriter.WriteStartDocument();
textWriter.WriteComment('Teste de comentario');
textWriter.WriteStartElement('Agenda');
textWriter.WriteStartElement('Contato');
textWriter.WriteStartElement('Nome');
textWriter.WriteString('Rodrigo');
textWriter.WriteEndElement();
textWriter.WriteStartElement('Fone');
textWriter.WriteString('123456789');
textWriter.WriteEndElement();
textWriter.WriteEndDocument();
textWriter.Close()
```

**Listagem 4.** Leitura de um arquivo XML utilizando XmlTextReader

```
var textReader: XmlTextReader := new XmlTextReader('banco.xml');
textReader.Read();
textReader.MoveToElement();
while textReader.Read() do begin
  if (textReader.NodeType = XmlNodeType.Text) then
  begin
    MessageBox.Show('Valor:' + textReader.Value);
  end;
end;
```

A **Listagem 3** demonstra o uso de XmlTextWriter na criação de um arquivo XML.

A classe XmlTextReader poderá ser utilizada para ler XML em um arquivo. Ela fornece análise direta de XML, além de disponibilizar a especificação XML 1.0 e as definições e padrões de nomenclaturas do World Wide Web Consortium (W3C), apresentando de forma simples o acesso ao fluxo de dados simbolizado em XML, ao invés de utilizar um modelo de objeto, como o modelo de objeto de documento (DOM) XML. XmlTextReader é utilizada quando é necessário acessar XML com dados sem a sobrecarga de um DOM, dessa maneira fornecendo um mecanismo mais rápido para leitura de XML. A classe XmlTextReader possui construtores diferenciados para especificar a localização dos dados XML. A **Listagem 4** indica a leitura de um arquivo XML utilizando XmlTextReader.

Para processar os dados XML, cada registro possui um tipo de nó que poderá ser determinado pela propriedade NodeType, além disso, as propriedades Value e Name indicam o nome do nó (os nomes do elemento e do atributo) e o valor (o texto do nó). A enumeração NodeType determina o tipo de nó utilizado ou referenciado.

## LINQ to XML

A linguagem LINQ utiliza os mesmos princípios do DOM para a modelagem, entretanto, oferece uma implementação mais intuitiva a partir de vários elementos, com os quais podem ser criados os nós, elementos, atributos, comentários e tudo o que puder fazer parte destes elementos.

Alguns métodos são utilizados diretamente no tratamento de arquivos XML, entre eles, destacam-se XName que é utilizado para criar uma identificação para o namespace do XML e seu uso é tanto para um elemento como para um atributo, o XNode que representa o nó dentro da árvore do documento XML e é usado como uma classe base para os elementos representados por XComment, XContainer, XDocumentType, XProcessingInstruction e XText; o XContainer que é um derivado de XNode, disponibilizando características e funcionalidades como subelementos de um nó XML, e é também a base para as pesquisas pelos nós XML e para XDocument e XElement; o XDocument responsável por representar um documento XML que contém os principais elementos do documento como a declaração do tipo do documento (XDocumentType), um elemento raiz (XElement) e podendo ter, opcionalmente, comentários (XComment); o XElement que é o tipo fundamental para construção de um documento XML devendo conter um XName e opcionalmente um ou mais atributos, podendo ter ou não algum conteúdo; o XAttribute utilizado para criar atributos que são elementos que possuem uma chave associada a um elemento; por último, o XComment usado para fazer comentários em um nó do documento XML.

O LINQ to XML permite realizar consultas LINQ em dados no formato XML, que tenham como origem arquivos de sistemas, Web Services através de uma chamada HTTP ou até mesmo em arquivos XML pré-existentes na memória. Algumas funcionalidades têm destaque como a opção de carregar XML a partir de

um arquivo ou streams, serealizar XML para arquivos ou streams, a criação de um XML do zero utilizando para isso estruturas funcionais, consultas em estruturas XML através de LINQ, a manipulação de estruturas XML na memória e a validação de estruturas XML através de XSD.

Vale destacar que realizando um comparativo entre o LINQ to XML e as classes presentes no .NET para manipulação de dados XML, na System.Xml (XmlReader/XmlWriter), a nova versão sem dúvida oferece mais recursos e facilidades para o manuseio de XML, além de ser mais eficiente, sendo que utiliza menos memória que o XmlDocument.

Através da **Listagem 5** é possível observar o uso de LINQ para a criação de um documento XML, semelhante ao que foi criado no exemplo demonstrado na **Listagem 1** e na **Listagem 3**. Observe a sintaxe declarativa de XDocument e a definição dos elementos através de XElement, como resultado, uma mensagem em tela será apresentada.

Para realizar consultas em LINQ, a cláusula from é utilizada para especificar a fonte de dados e uma variável é usada para referenciar cada elemento da fonte de dados. A cláusula select é usada para fazer projeções, permitindo especificar o tipo e o formato dos elementos do resultado da consulta. Além disso, o recurso de tipos anônimos também está presente, permitindo construir uma projeção do resultado de uma consulta LINQ sem a necessidade de se criar um novo tipo manualmente.

Outras opções estão disponíveis como as cláusulas let, orderby e where, essa última permite filtrar elementos da fonte de dados baseada em uma ou mais expressões booleanas e separadas por operadores lógicos. A **Listagem 6** indica uma consulta LINQ to XML utilizando um arquivo XML para realizar uma projeção de dados e sua posterior visualização através de mensagem na tela do usuário do aplicativo.

## Web Services

O Delphi Prism disponibiliza o uso de Web Services em aplicativos do tipo Web, como por exemplo, uma Web Application com a adição de um novo item como mostra a **Figura 1**. As funcionalidades descritas pelo Web Service estão presentes neste arquivo que irá disponibilizar os Web Methods através de WSDL que será instanciado por um aplicativo por meio de uma Web Reference.

A **Listagem 7** demonstra o código para um WebMethod básico chamado de HelloWorld. Após a definição de um Web Service, este poderá ser consumido por aplicativos de diferentes plataformas, incluindo aplicativos que contenham formulários Windows Forms, aplicações nativas Delphi Win32, PHP, AJAX, .NET, Java etc. No caso do .NET deverá ser adicionada uma referência através de Add Service Reference, incluindo na seção uses o namespace da referência criada (também conhecida por classe Proxy). Para fins de testes o código da **Listagem 8** possibilita executar o método HelloWorld através de um clique do mouse em um botão.

A tecnologia trabalha com a transferência de dados através do protocolo HTTP, encapsulados pelo protocolo SOAP (Simple Object Access Protocol), fazendo com que assim exista um isolamento entre o serviço, as tarefas a serem executadas e até mesmo o banco de dados. Assim sendo, para existir uma comunicação com qualquer Web Service é necessário uma implementação do protocolo SOAP para comunicação e que deverá ser determinado pelos princípios de regras descritas pelo W3C. Geralmente servidores SOAP são codificados utilizando-se servidores HTTP, embora isto não seja uma restrição

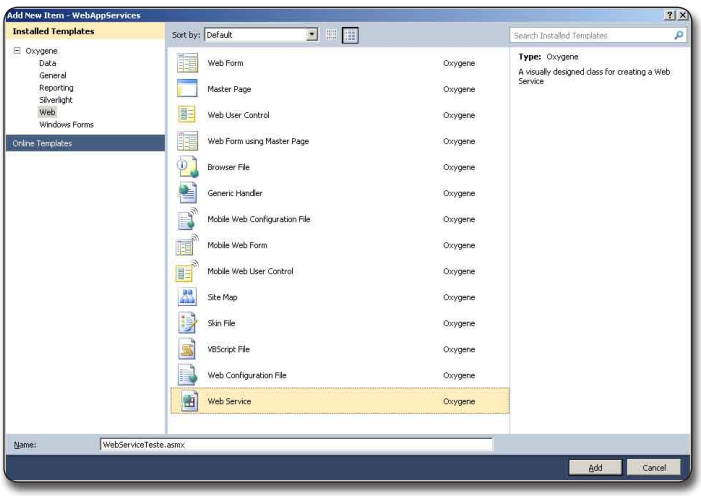



**Figura 1.** Criando um Web Service no Delphi Prism

**Listagem 5.** Criando um arquivo XML utilizando LINQ to XML

```
var agenda: System.Xml.Linq.XDocument := new XDocument(new XDeclaration
('1.0', nil, nil),
 new XElement('Agenda', new XElement('Contato', new XElement('Nome','Rodrigo'),
  new XElement('Fone','123456789') ),
  new XElement('Contato', new XElement('Nome','Thiane'),
  new XElement('Fone','234567890') ),
  new XElement('Contato', new XElement('Nome','Daniela'),
  new XElement('Fone','554567890') )
  ));
MessageBox.Show(agenda.ToString());
```

**Listagem 6.** Consulta à fonte de dados utilizando LINQ to XML

```
  var arquivo: XElement := XElement.Load('.\banco.xml');
  var contatos := from b in arquivo.Elements('Contato')
  select new class (NomeContato := b.Element('Nome'),
   FoneContato := b.Element('Fone'));
  for each dados in contatos do
  begin
    MessageBox.Show('Nome:' + dados.NomeContato + '  Fone:' + dados.
    FoneContato);
  end;
```

**Listagem 7.** Criando um WebMethod em Delphi Prism

```
[WebMethod]
method WebServiceTeste.HelloWorld(): string;
begin
 result := 'Hello World';
end;
```

**Listagem 8.** Código para executar um método de um Web Service

```
method MainForm.button5_Click(sender: System.Object; e: System.EventArgs);
begin
 var web: WebServiceTesteSoapClient := new WebServiceTesteSoapClient();
 MessageBox.Show(web.HelloWorld());
end;
```

para funcionamento do protocolo. As mensagens SOAP são documentos XML que aderem a uma especificação fornecida pelo órgão W3C.

O primeiro esforço do desenvolvimento do SOAP foi implementar RPC sobre XML. RPC (Remote Procedure Call) é o tipo de protocolo para chamada remota de procedimentos em qualquer lugar da rede ou uma chamada de função para o método de transferência de controle de parte de um processo para outro, permite a divisão de um software em várias partes. RPC pode ser utilizado sobre diferentes protocolos de transporte, pois não especifica como a mensagem é enviada, somente lê e interpreta. As camadas de transporte que o RPC implementa são a TCP e a UDP. Além disso, RPC e Sockets têm a função de implementar numa aplicação as funções de rede, mas fazem isto de maneiras diferentes.

## Pesquisa de objetos na memória

Nem tudo pode ser acesso por SQL, existirão casos em que será necessário trabalhar com alguns dados que existem apenas na memória. Uma das situações onde isto poderá acontecer é quando se tem uma lista de dados e se deseja validar uma entrada que foi feita pelo usuário.

Muitas vezes, por conhecer poucas ferramentas para pesquisa de dados na memória acaba-se escrevendo uma quantidade desnecessária de código que justamente por ser desestruturado ou pouco planejado, acaba sendo repetido em várias partes do programa, o que gera confusão e perda de tempo. Pesquisas de dados em memória normalmente envolvem loops (For) e blocos condicionais (If).

As listas genéricas estão presentes a partir da versão 2.0 do framework. Além de oferecer uma maneira simples de trabalhar com coleções dinâmicas, também possuem um grande número de recursos para realizar pesquisas nos dados contidos dentro da mesma, inclusive através de LINQ.

As listas genéricas têm tamanho dinâmico, podendo ser alterado facilmente através da inclusão ou exclusão de elementos. Também podem ser facilmente gerados a partir de um array tradicional, bastando para isso passar o array existente para o construtor como descrito pela **Listagem 9**. É definida uma lista e adicionado alguns valores à sequência de dados. Outras opções envolvendo listagens poderão ser utilizadas como pesquisa, ordenação e até exclusão de informações da lista.

Para realizar consultas em arrays e listas genéricas ou não armazenadas na memória, poderá ser utilizado o LINQ to Objects que corresponderá à forma utilizada para pesquisa em coleções de objetos. O código para a consulta da lista presente na **Listagem 9** poderá ser semelhante ao da **Listagem 10**.

**Listagem 9.** Código para definição de uma lista

```
var lista: List<System.Int32> := new List<System.Int32>(array of System.
Int32([1, 2, 3, 4]));
lista.Add(10);
lista.Add(15);
```

**Listagem 10.** Código para consulta utilizando LINQ to Objects

```
var ResultadoDaPesquisa := from str in lista where str = 15 select str;
 for each dados in ResultadoDaPesquisa do
 begin
   MessageBox.Show('Resultado:' + dados.ToString());
 end;
```

**Listagem 11.** Código para utilizar operador de consulta single

```
 var nomes: List<System.String> := new List<System.String>;
 nomes.Add('Rodrigo');
 nomes.Add('Teste');
 var nome: System.String := nomes.Single(n -> n = 'Rodrigo');
 MessageBox.Show(nome);
```

**Listagem 12.** Código para utilizar operador de consulta where

```
 var nomes: List<System.String> := new List<System.String>;
 nomes.Add('Rodrigo');
 nomes.Add('Marcela');
 nomes.Add('Roberta');
 for each resultado in nomes.Where(nome -> nome.StartsWith('Ro')) do
 begin
   MessageBox.Show('Os nomes encontrados são:' + resultado);
 end;
```

**Listagem 13.** Uso de orderby em uma consulta de dados

```
 var nomes: List<System.String> := new List<System.String>;
 nomes.Add('Rodrigo');
 nomes.Add('Marcela');
 nomes.Add('Roberta');
 nomes.Add('Aline');
 for each nome: System.String in nomes.OrderBy(VariaveldeTeste ->
 VariaveldeTeste) do
 begin
   MessageBox.Show(nome);
 end;
```

## Lambda Expressions

Muitos Query Operators, por padrão, podem utilizar delegates (uma forma segura de ponteiro para função) para processar elementos individuais que operam em uma sequência. As Lambda Expressions podem trabalhar em conjunto com os Query Operators de consulta para representar delegates, e são uma forma abreviada para a criação do seu conjunto de implementação. Em resumo, elas fornecem uma sintaxe funcional para escrever Anonymous Methods.

Na **Listagem 11** é demonstrada a sintaxe básica do Query Operator "Single" utilizando Lambda Expression, que retorna o valor da propriedade que for verdadeira para o critério. Se nenhuma propriedade estiver em conformidade com o critério de consulta, é realizada uma exceção de código.

Observe que esta sintaxe é bastante poderosa e concisa. Note que a ideia é a mesma dos Anonymous Types, e nesse exemplo, **n** é o nome dado à variável utilizada para iterar na lista de nomes. Além disso, poderia ter sido dado outro nome a esta variável, ficando isto a critério do desenvolvedor.

Da mesma forma, o uso do operador where possibilita realizar consultas semelhantes ao Where da SQL, realizando uma filtragem dos dados. Observe na **Listagem 12** que o critério para consulta é que o nome comece com as letras "Ro", e nesse exemplo os resultados retornados em uma mensagem para o usuário serão "Rodrigo" e "Roberta".

Além disso, um operador bastante útil é o Query Operator "OrderBy", que tem como funcionalidade ordenar os valores em ordem crescente. É um exemplo claro de outra operação similar ao que existe na SQL. Note na **Listagem 13** que ainda é necessária a adição de um critério em sua assinatura, mesmo que vazio. Nesta listagem os dados ordenados serão apresentados através de mensagem para o usuário, e seu resultado será na sequência: "Aline", "Marcela", "Roberta" e "Rodrigo".

## Expressões Regulares

As expressões regulares são projetadas e otimizadas para a manipulação de texto, compostas por dois tipos básicos: caracteres de texto literal e caracteres especiais. Podem ser usadas para validar entradas de dados que devem obedecer um padrão, em pesquisas e comparações.

Para fazer uso de uma expressão regular, basta adicionar na cláusula uses do seu projeto a referência a "System.Text.RegularExpressions" o que possibilitará o uso direto da funcionalidade regex semelhante ao que ocorre na **Listagem 14**, onde é definido um formato padrão de entrada de dados para um controle TextBox no formato de um CPF ao estilo "123.456.789-00". Observe que para dados fora deste padrão, uma mensagem em tela será mostrada ao usuário indicando que está apresentando um formato incorreto, situação bastante útil para validar informações e não permitir a sequência das operações do aplicativo enquanto a condição não for satisfeita.

As expressões regulares apresentam fortes recursos para pesquisa, inclusive pesquisa avançada de textos. Na **Listagem 15** é definida uma expressão regular para realizar a pesquisa de determinada sequência de caracteres que representem uma espécie de endereço de internet, por exemplo "http://algumacoisa.outracoisa". Se for informado o texto "observe o link http://www.devmedia.com.br e veja todas as novidades e lançamentos das revistas." a expressão regular retornará como resultado o link que é a parte do texto que satisfaz suas condições.

A variável resultado do tipo Match executa a pesquisa em uma string, o resultado é armazenado realizando uma estrutura de repetição e continuando a pesquisa através de NextMatch. Deve ser observado que a mensagem presente na estrutura de repetição indicará o valor do texto encontrado que satisfaça a condição da expressão regular, representado por uma string, para o usuário através de uma mensagem. Ao informar um texto em um controle TextBox poderá ser verificado o seu comportamento e extração das informações pertinentes a expressão regular.

## ADO.NET

O .NET Framework inclui uma tecnologia de acesso a dados, o ADO.NET, que é uma evolução do antigo ADO (ActiveX Data Objects), sendo projetado principalmente para ser utilizado em aplicações distribuídas na Web. Representa uma forma de manipulação de dados baseado em DataSets desconectados, disponibilizando um completo e poderoso modelo de programação para acesso a dados utilizando, por padrão, o formato XML para transmissão de dados.

**Listagem 14.** Uma expressão regular para validar um formato de CPF

```
var expressaoCPF: Regex := new Regex('^\d{3}\.\d{3}\.\d{3}-\d{2}');
if not expressaoCPF.IsMatch(textBox1.Text) then
begin
  MessageBox.Show('CPF com formato inválido!');
end;
```

**Listagem 15.** Uma expressão regular para consultas em textos

```
var expressao: Regex := new Regex('(http)://([_a-z\d\-]+(\.[_a-z\d\-]+)+)');
var resultado: Match;
resultado := expressao.Match(textBox1.Text);
while resultado.Success do
begin
  MessageBox.Show(resultado.Value);
  resultado := resultado.NextMatch()
end;
```

**Listagem 16.** Utilizando um DataSet para exibir dados em um DataGridView

```
var conexao: SqlConnection := new SqlConnection('Data Source=APACHE\
SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True');
var adapter: SqlDataAdapter := new SqlDataAdapter('select * from
Employees', conexao);
var dsEmployees: System.Data.DataSet := new System.Data.DataSet();
if (conexao.State = System.Data.ConnectionState.Closed) then
begin
  conexao.Open();
end;
adapter.Fill(dsEmployees);
conexao.Close();
if (dsEmployees.Tables[0].Rows.Count>0) then
begin
  dataGridView1.DataSource := dsEmployees.Tables[0];
end;
```

O ADO.NET também garante a escalabilidade requerida por aplicações Web que compartilham dados. Essas aplicações devem servir a dezenas, centenas e até milhares de usuários ao mesmo tempo. O ADO.NET não retém locks e conexões ativas com os bancos de dados, as quais podem consumir recursos que são limitados. Isso possibilita aumentar o número de usuários com pequenos aumentos na demanda de recursos do sistema.

Pode ser utilizado por qualquer linguagem compatível com o .NET, inclusive o Delphi Prism. O ADO.NET é um modelo de objetos composto por diversas classes, que podem ser divididas em dois grandes grupos: Data Classes, que são capazes de conter dados recuperados de bancos, e os Managed Providers, que fazem o acesso aos dados. Com o primeiro grupo, pode-se trabalhar de forma desconectada do banco, e com o segundo é trabalhado de maneira conectada.

Os Managed Providers são utilizados para acessar os dados diretamente no banco. Eles estão divididos em famílias e cada família é responsável por acessar um tipo de fonte de dados, o acesso a fonte de dados é bastante variado podendo englobar as mais utilizadas entre elas, apenas como exemplo, SQL Server, OLE DB, Oracle, ODBC entre outras. Para a sua utilização, no código fonte desejado, deverá ser indicado o respectivo namespace do provider.

A **Listagem 16** demonstra o uso de um DataSet para exibir dados em um DataGridView, para que isto seja possível, é utilizado como exemplo o Banco de Dados Northwind presente no SQL Server e

realizada uma consulta à tabela Employees. Observe que logo após realizar o Fill para o DataSet, a conexão é fechada e será trabalhado em modo desconectado, na verdade as linhas que abrem e fecham a conexão com o banco de dados podem ser comentadas sem que interfira no resultado final para o DataGridView.

Sem dúvida o principal componente do ADO.NET é o DataSet. Ele pode ser utilizado com qualquer um dos providers para ADO.NET, representando uma estrutura de dados em memória. Aqui o termo "memória" se refere ao fato de que poderá ser criada uma consulta a uma tabela do banco de dados, extrair informações e usá-las após a conexão ser fechada. Por esse motivo, aplicações que usam esse componente são conhecidas como "desconectadas".

Enquanto um DataReader exige que uma conexão esteja ativa para que uma leitura de dados seja feita, um DataSet pode obter seus dados uma única vez e guardá-los em memória para uso posterior. Isso garante ainda mais a escalabilidade das aplicações em Delphi Prism. Dependerá apenas da decisão de qual será a melhor forma a ser utilizada no aplicativo levando em conta cada situação de uso de quando será melhor usar um DataReader ou um DataSet.

Na **Listagem 17** é possível verificar o uso de um DataReader para preencher dados em um controle ListBox. Observe que neste exemplo é necessário que a conexão com o banco de dados permaneça aberta para que a leitura dos dados ocorra de forma correta. No ListBox será mostrado o campo FirstName e Country de um registro da tabela Employees. Um DataReader é responsável por fazer a leitura dos dados retornados por uma consulta SQL, usando um cursor de dados. Ele é rápido por ser unidirecional, sua navegação pelos registros é feita de forma sequencial (forward-only). Basicamente um registro é lido, é realizada alguma operação com ele (no exemplo os dados serão exibidos em um ListBox) e ocorrerá a navegação para o próximo registro. Além disso, apresenta a característica de ser de somente-leitura, não sendo possível alterar registros de um DataReader e não realiza cache após um registro ser lido, ele é descartado da memória.

**Listagem 17.** Utilizando um DataReader para exibir dados em um ListBox

```
var conexao: SqlConnection := new SqlConnection('Data Source=.\SQLEXPRESS;
Initial Catalog=Northwind;Integrated Security=True');
var comando: SqlCommand := new SqlCommand('select * from employees', conexao);
if (conexao.State = System.Data.ConnectionState.Closed) then
begin
  conexao.Open();
end;
var dr : SqlDataReader;
dr := comando.ExecuteReader();
while (dr.Read()) do
begin
  listBox1.Items.Add(dr['FirstName'].ToString() + ' - ' + dr['Country'].ToString());
end;
conexao.Close();
```

## Conclusão

Seja qual for o tipo de aplicativo desenvolvido, Web, Mobile, Desktop, as principais tarefas do dia a dia envolverão de alguma forma a manipulação de documentos XML, acesso a dados e uso de Web Services. Em todos esses cenários, as expressões Lamda e LINQ estarão presentes. Este artigo mostrou na prática como você desenvolvedor Delphi Prism pode tirar proveito dos recursos do .NET Framework para realizar estas tarefas com o mínimo de codificação possível.

**Rodrigo Araujo**
*rodrigo.arsi@gmail.com*
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

**Delphi Prism**
*http://www.embarcadero.com/products/delphi-prism*

**Visual Studio 2010**
*www.microsoft.com/VisualStudio*

**Dê seu feedback sobre esta edição!**

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

***www.devmedia.com.br/clubedelphi/feedback***

# Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

**Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.**

TODAS as vídeo aulas que você sempre quis ver a partir de

**R$ 0,99**

**ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!**

**Saiba mais sobre o Sistema de Créditos!**

**www.devmedia.com.br/creditos**

DevMedia

# Hospede seus sites de forma simples, rápida e ilimitada.

## Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

**HOSPEDIX**
HOSPEDAGEM PROFISSIONAL
**www.hospedix.com.br**

**30 DIAS GRÁTIS**

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

**RVSTMEDIA**